AD-A257 238

Technical Report

CMU/SEI-92-TR-25
ESC-TR-92-025

Carnegie-Mellon University
Software Engineering Institute

Software Measures and the Capability Maturity Model

John H. Baumert
Mark S. McWhinney

September 1992

DTIC
ELECTE
OCT 27 1992
D
E

DISTRIBUTION STATEMENT A
Approved for public release
Distribution unlimited

416208

92-28198
391
pg

Carnegie Mellon University does not discriminate and Carnegie Mellon University is required not to discriminate in admission, employment or administration of its programs on the basis of race, color, national origin, sex or handicap in violation of Title VI of the Civil Rights Act of 1964, Title IX of the Educational Amendments of 1972 and Section 504 of the Rehabilitation Act of 1973 or other federal, state or local laws, or executive orders.

In addition, Carnegie Mellon University does not discriminate in admission, employment or administration of its programs on the basis of religion, creed, ancestry, belief, age, veteran status, sexual orientation or in violation of federal, state or local laws, or executive orders. While the federal government does continue to exclude gays, lesbians and bisexuals from receiving ROTC scholarships or serving in the military, ROTC classes on this campus are available to all students.

Inquiries concerning application of these statements should be directed to the Provost, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, Pa. 15213, telephone (412) 268-6684 or the Vice President for Enrollment, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, Pa. 15213, telephone (412) 268-2056.

Technical Report
CMU/SEI-92-TR-25
ESC-TR-92-025
September 1992

Software Measures and the Capability Maturity Model

John H. Baumert

Software Process Measurement Project
Resident Affiliate, Computer Sciences Corporation

Mark S. McWhinney

Software Process Measurement Project

Accesion For
NTIS
DTIC
Unannounced
Justification
By
Distribution /
Availability Codes
Dist | Avail and/or Special
A-1

Approved for public release.
Distribution unlimited.

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This technical report was prepared for the

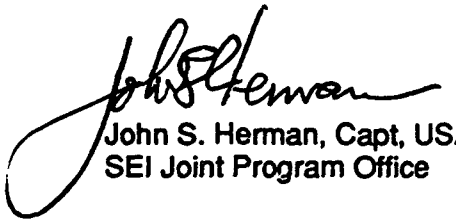
SEI Joint Program Office
ESC/AVS
Hanscom AFB, MA 01731

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER



John S. Herman, Capt, USAF
SEI Joint Program Office

The Software Engineering Institute is sponsored by the U.S. Department of Defense.
This report was funded by the U.S. Department of Defense.

Copyright © 1992 by Carnegie Mellon University.

This document is available through the Defense Technical Information Center. DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center, Attn: FDRA, Cameron Station, Alexandria, VA 22304-6145.

Copies of this document are also available through the National Technical Information Service. For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161.

Copies of this document are also available from Research Access, Inc., 3400 Forbes Avenue, Suite 302, Pittsburgh, PA 15213.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

Table of Contents

List of Figures

List of Tables

Acknowledgments

The authors recognize and thank the individuals who generously contributed their time to review and comment on earlier drafts of this technical report. We feel that their comments and suggestions have resulted in a better document. These people are:

John Beck
AT&T Bell Laboratories

Mary Busby
IBM Corporation

David Card
Computer Sciences Corporation

Anita Carleton
Software Engineering Institute

Mary Beth Chrissis
Software Engineering Institute

Bill Curtis
Software Engineering Institute

William Florac
Software Engineering Institute

John Harding
Bull HN Information Systems, Inc.

Jim Hart
Software Engineering Institute

Jeffrey Heimberger
The MITRE Corporation

Watts Humphrey
Software Engineering Institute

Wolfhart Goethert
Software Engineering Institute

Robert Grady
Hewlett-Packard

Shari Lawrence Pfleeger
The MITRE Corporation

Donald McAndrews
Software Engineering Institute

Richard Mendez
AT&T Bell Laboratories

Tim Olson
Software Engineering Institute

Robert Park
Software Engineering Institute

Mark Paulk
Software Engineering Institute

Jane Siegel
Software Engineering Institute

Patricia Van Verth
Canisius College

Dave Zubrow
Software Engineering Institute

We also thank Lori Race for her outstanding secretarial support and patience during the preparation of this technical report.

Software Measures and the Capability Maturity Model

Abstract. This document describes a set of software measures that are compatible with the measurement practices described in the Capability Maturity Model for Software. These measures, in the form of software indicators, cover thirteen different categories that include progress, effort, cost, and quality. Each indicator category contains example figures which illustrate behavior that may occur on a project. The text provides users with tips on how to use these figures or similar ones on their projects. Project software managers and software engineering process groups can use these indicators during the software development life cycle to gain insight into the software development process and software process improvement activities. The indicators chosen have been successfully used on projects in the software industry.

1. Introduction

The Software Process Measurement Project at the Software Engineering Institute (SEI) promotes the use of measurement in improving the management of software development and the acquisition of software. The project has worked and continues to work with representatives from industry, government, and academia to develop basic definitions of software measures and a measurement process that can be used to systematically and repeatedly measure software development progress, products, and processes. Four documents that describe these measures have been released [Florac 92], [Goethert 92], [Park 92], and [Rozum 92]. This document complements these documents by providing a set of software measures in the form of indicators that are compatible with the measurement practices of the Capability Maturity Model for Software (CMM) [Paulk 91], [Weber 91]. In this document, *indicator* is used to mean a representation of measurement data that provides insight into software development processes and/or software process improvement activities. A measure quantifies a characteristic of an item; an indicator may use one or more measures. For example, an indicator may be the behavior of a measure over time or the ratio of two measures.

1.1. Objectives

The goal of this document is to provide a comprehensive and cohesive set of indicators that is consistent with the key practices in the CMM. This document describes and serves as a reference manual for that set of software indicators.

The objectives of this document are to:

1. Provide software indicators that are consistent with the key practices of the CMM.
2. Provide software indicators that address the measurement-related goals of the CMM key process areas.
3. Provide information on the use of the indicators.
4. Provide organizations that have no measurement program with indicators that can be used when such a program is started.
5. Provide organizations that have a measurement program with indicators that can be used for comparison with their programs.

The first two objectives address the relationship between the indicators and the CMM. They also show that measurement is a critical technology which should be practiced at each maturity level. The third objective emphasizes that it is not sufficient to provide a list of software indicators; guidance in their use is at least as important.

The last two objectives indicate that the document is written to provide useful information to organizations with or without a measurement program. An organization without a measurement program may use this document to determine what measures can be used and how to use the indicators. An organization with a measurement program may use this document to compare their measures with the indicators discussed in this document and add or delete measures accordingly.

A project or organization is not required to use the indicators discussed in this document. The indicators represent a set that is consistent with the CMM key practices but do not represent a universal set. Other indicators may exist that are also compatible and useful within the framework of the CMM.

1.2. Scope

The indicators track both software products and software development processes with the emphasis on the processes. They are not identified as process or product indicators because often one indicator can point to either a process or product characteristic. Also, product characteristics such as maintainability and portability are not addressed specifically as there are no standard measures of these characteristics. Occasionally, they can be inferred from the indicators. For example, product reliability and maintainability can be inferred from the numbers of defects found in the product.

Indicators are given for software development activities only. Due to the limited space available, only the life-cycle stages from requirements through testing are addressed so that a comprehensive indicator set for software development could be recommended. Information on the use of the indicators is focused on software development. Many of the indicators can also be used in operations and maintenance, but no claims of completeness are made with respect to indicators for this life-cycle stage.

The implementation of a broad-scale measurement program is also not discussed. Such suggestions can be found in a number of sources [DeMarco 82], [Grady 87], [Jones 91], [Pfleeger 89], [Kuntzmann 92]. The number of indicators and sample graphs in this document is relatively large and can discourage an organization from establishing a measurement program. An organization needs to weigh its measurement goals against its resources to determine the structure and content of its measurement program. Then it can select indicators from this document or develop its own that satisfy those goals.

1.3. Audience

This document is written for the project software manager and the software engineering process group (SEPG). The example graphs and interpretations discussed in this document provide information that is useful to a project software manager or an SEPG. Others may use the same or similar graphs but with a different focus and interpretation. Figure 1.3-1 shows potential users of these software indicators.

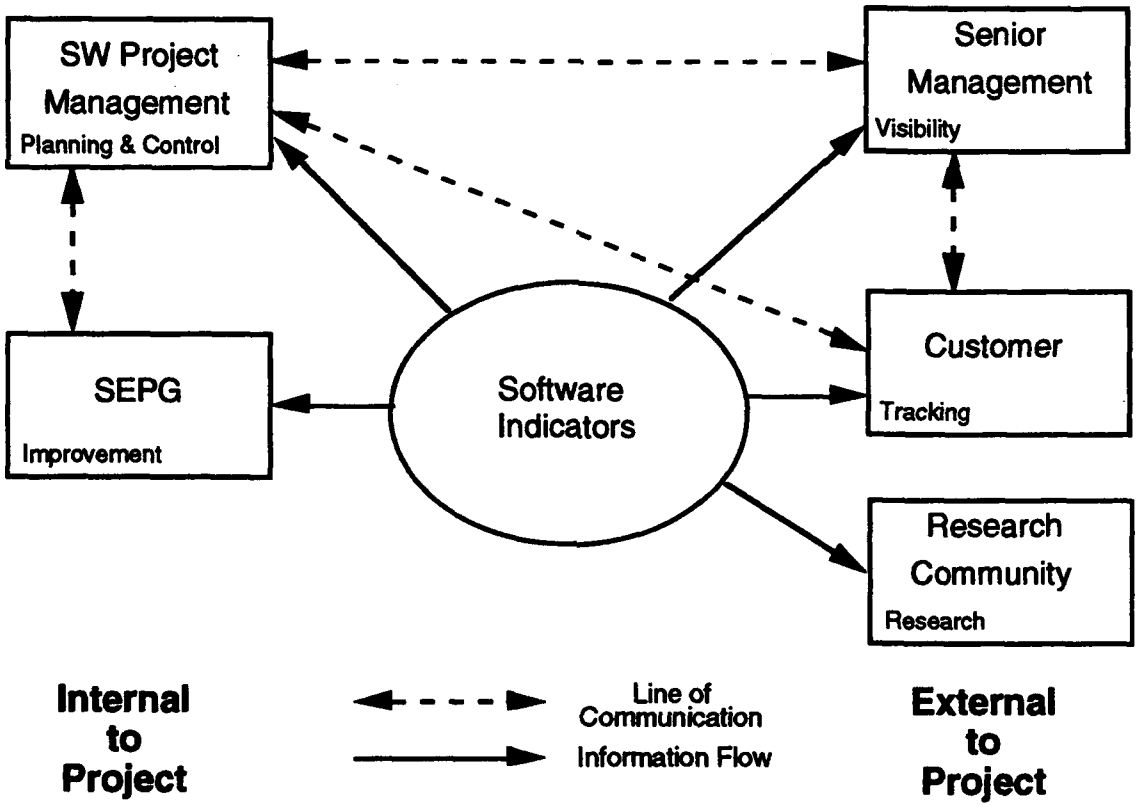


Figure 1.3-1. Software Measurement Users

All levels of project management can use the indicators for planning and controlling project cost, schedule, quality, risks, and resources. In addition, the indicators allow project managers to make informed decisions and to discuss the status and performance of the project with senior management and customers.

The SEPG can use the indicators to improve the project as well as the organization, to determine the quality of the products and processes of the project, and to monitor the effectiveness of improvements put into place.

Senior management wants visibility into the cost, schedule, quality, risks, resources, and improvements occurring on each project under its control, but is usually interested in the project as a whole and not the details. The indicators provide this visibility as well as a common ground for discussions with the project manager and the customer. The indicators also provide senior management with information about the capability of the organization as a whole.

The customer can use the indicators to track the project cost, schedule, risks, and quality. The research community is interested in obtaining a consistently defined set of accurate data and can use the measures that formed the indicators.

1.4. How to Use This Document

The indicators discussed in this document are descriptive and are not intended to be prescriptive. That is, they are recommended and not mandated. They form a cohesive set that can serve as the nucleus of a measurement program within an organization. Each organization is responsible for determining its goals, devising its own process improvement program, and determining the measurements that best fit its program.

The indicators are discussed in relation to specific maturity levels. However, the use of the indicators from a particular maturity level will not guarantee that an organization is performing at that maturity level. Measurement is only part of the CMM. An organization must also perform other activities, such as training, establishing policies, verifying implementation, and so forth.

Within the CMM, a higher maturity level encompasses the key practices of the lower levels. In general, the same is true of the indicators discussed in this document. The indicator categories are the same for all maturity levels. Additional indicators are placed in these categories as the organization matures. Indicators from lower levels may also change as the organization matures. If this occurs, the discussion of the indicator is appropriate at the higher maturity level. For example, statistical process control techniques are applied at the Managed Level to certain lower level indicators. The more simplistic indicators at lower maturity levels may not be used at the higher levels, but the measures used to derive them are available for use at the higher maturity levels.

The definitions of input data to the indicators do not change from maturity level to maturity level. This allows the use of the indicators at different maturity levels. The

inputs remain constant, but the analysis of the input data may change. Again, this allows the organization to use historical data to predict performance at higher maturity levels.

Although each indicator is discussed separately, it should *NOT* be used or interpreted in isolation. Indicators are used in conjunction with other indicators to get a more complete picture of what is occurring on the project and within the organization. For example, the cost indicator is related to the effort and progress indicators. All three indicators should be used and interpreted together. In addition, the cost of a project can increase or decrease depending on the number of requirements changes. Therefore, the project software manager uses the requirements stability indicator along with the cost indicator. Similarly, the effort indicator is related to the cost, progress, and training indicators.

The level of detail given for the indicators at the Repeatable and Defined Levels is greater than at the Managed and Optimizing Levels. This is an artifact of the measurement discussion in the CMM. Knowledge of the Managed and Optimizing Levels is incomplete. Most work in software measurement is concerned with issues that are covered by key process areas at lower levels of maturity. As organizations mature, experience with indicators at higher levels will increase, and consequently more detail can be added to the discussion of the indicators at the higher levels.

If an organization wishes to use the indicators discussed in this document to establish or modify a measurement program, it is recommended that the organization review the indicators across all maturity levels to derive a comprehensive and coherent program, not a program pieced together one maturity level at a time. This is particularly important with respect to the inputs to the indicators. The data not only need to be accurate, but they also need to be complete and consistent across all maturity levels.

Many of the indicators involve monitoring a quantity over time. The reporting period shown in an example graph is arbitrarily selected. In practice, the frequency of reporting or monitoring is dependent on the size and duration of the project and is determined by the project. For large projects or multi-year projects, a monthly report is appropriate. Weekly or biweekly reporting periods are typical of projects with shorter duration.

The reporting and monitoring frequency is also dependent on the type of indicator and for whom the report is prepared. For example, lower-level software managers would monitor progress on a weekly basis for the work under their control but may report progress on a monthly basis to the project software manager. Likewise, software quality assurance personnel would monitor results of audits on a biweekly or monthly basis, but trouble reports on a weekly basis.

1.5. Document Organization

This document is divided into seven chapters. Chapter 2 provides background material and the approach used to derive the software indicators that are listed in Chapter 3 and discussed in detail in Chapters 4 through 7.

The appendices contain a list of acronyms and definitions of terms used in the document, a mapping of software measurement references in the CMM key practices to indicator categories, and a mapping of the software indicators to software measurement references in the CMM key practices.

Each indicator is discussed in a separate section, that is, as a stand-alone entity. The discussion of each indicator section in Chapters 3 through 7 has the following format:

- Objective of the Indicator—the purpose of the indicator
- Indicators—a listing of the indicators within the indicator category
- Key Process Area Goals Addressed—the CMM key process area goals that are the focus of the indicators
- Life-Cycle Stage—where in the life cycle the indicator should be used
- Users—people who would be the most likely to find the indicators useful or who would use the indicators most often
- Users' Questions—examples of the types of questions users could answer with the indicator
- Input—the data items or information needed to derive the indicators
- Interpretation—information on the indicators
- Sources—references to material(s) that further explain or describe the indicators

Figure 1.4-1 shows the relation of the indicators to the CMM maturity level and to several of the discussion items. Graphs are used to illustrate the use of the indicators. Unless noted, they are not based on real data. The actual appearance of a graph for a specific project or organization depends on the implementation of the concepts presented here and the specific graphing tool used.

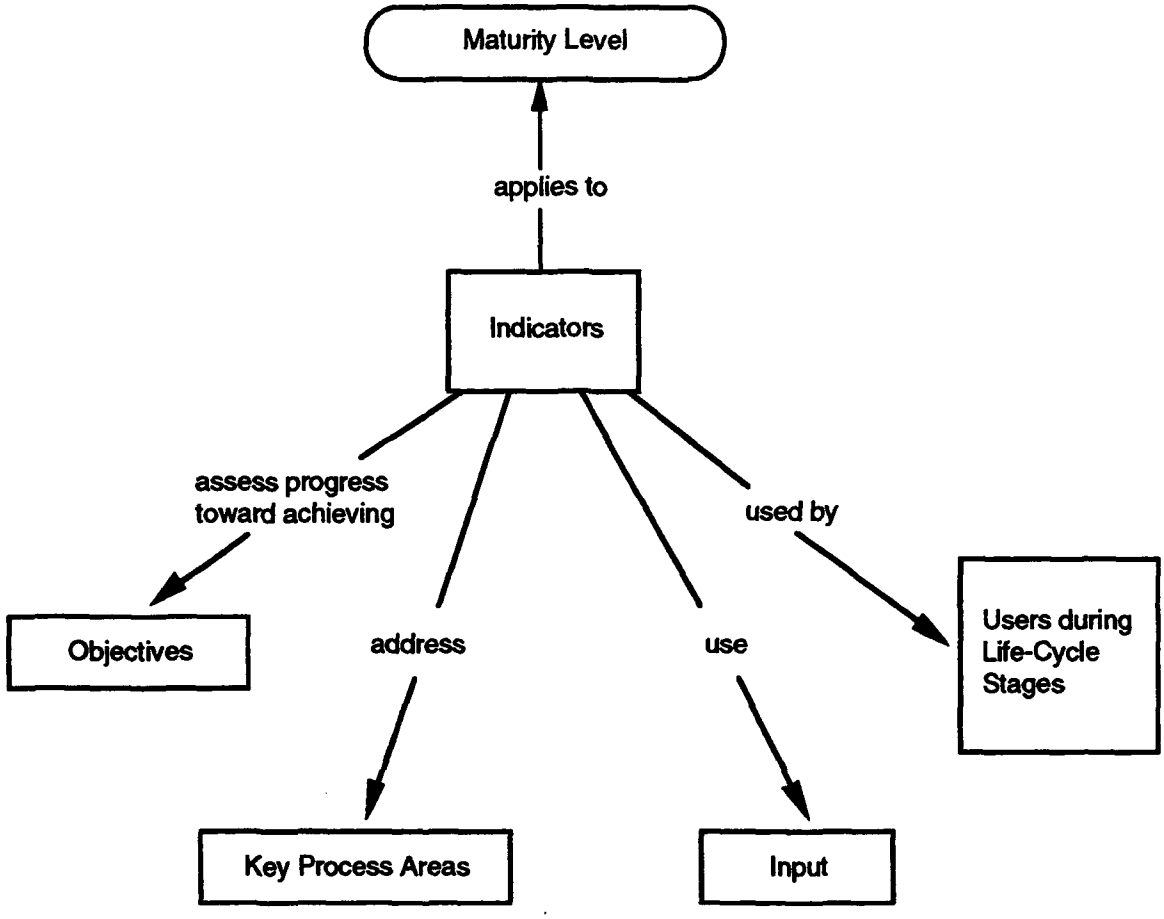


Figure 1.4-1. Relationship of Indicators to CMM Maturity Level and Indicator Discussion Topics

2. Background and Approach

This chapter provides background information that serves as a basis for the integration of the software measures and the Capability Maturity Model for Software (CMM) and describes the approach used in the integration of the two.

2.1. Software Measurement

Organizations with successful measurement programs report the following benefits:

- Insight into product development
- Capability to quantify tradeoff decisions
- Better planning, control, and monitoring of projects
- Better understanding of both the software development process and the development environment
- Identification of areas of potential process improvement as well as an objective measure of the improvement efforts
- Improved communication

However, many of the potential benefits that an organization can derive from a sound measurement program is often not achieved due to a half-hearted commitment by managers to a measurement program. The commitment cannot be just a policy statement; it must be total commitment. The policy must be followed with the allocation of resources to the measurement program. This includes allocating staff as well as tools.

It is important that an individual or group be assigned responsibility for the measurement program. This group identifies the measures to be collected, establishes training in measurement, monitors the consistency of the measures across the projects and throughout the organization, and can help projects initiate measures. This group can also provide composites of historical data that managers can use in planning and monitoring of projects.

The human element in a measurement program should not be taken lightly. Success and failure are tied to people. All staff members need to see the benefits of the measurement program and understand that the results will not be used against them. The focus of any measurement program is on the process and the product, not people. Jones has a section, entitled "The Sociology of Software Measurement," in the book in which he discusses data confidentiality, the use of data for staff performance targets, measuring one-person projects, management information systems vs. systems software, and measurement expertise [Jones 91].

Grady and Caswell discuss their experiences in establishing a measurement program at Hewlett-Packard [Grady 87]. All aspects of measurement are discussed: what to

measure, how to convince management and staff of the benefits, validity of data, the need for a database, training and tools, and so forth. Their practical experience is invaluable.

Rifkin and Cox summarize the current state of measurement practice in eleven divisions of eight organizations that have the reputation of having excellent measurement practices [Rifkin 91]. They describe the patterns that emerged at a consolidated lessons-learned level but also provide informative material in the individual case studies.

2.2. The Capability Maturity Model

The CMM is designed to provide organizations with guidance on how to gain control of their process for developing and maintaining software and how to evolve toward a culture of software excellence. It does this by serving as a model against which an organization can determine its current process maturity and by identifying the few issues most critical to software quality and process improvement.

This section provides a high-level overview of the CMM and its structure. Details are provided by Paulk and Weber [Paulk 91], [Weber 91]. Additional information on maturity levels and software process is given by Humphrey [Humphrey 89].

The CMM contains five levels of software process maturity: Initial, Repeatable, Defined, Managed, and Optimizing (see Figure 2.2-1). An organization at the Initial Level is characterized as one without a stable environment for developing and maintaining software. Few stable software processes are in place, and performance can only be predicted by individual, rather than organizational, capability.

An organization at the Repeatable Level has installed basic software management controls; that is, stable processes are in place for planning and tracking the software project. Project software managers track software costs, schedules, and functionality; problems in meeting commitments are identified when they arise. Software configuration management procedures are used to baseline and control software requirements. Project standards exist, and the software quality assurance group ensures that they are followed. In essence, there is a stable, managed, working environment.

An organization at the Defined Level has a standard process for developing and maintaining software across the organization. The software engineering and software management processes are integrated into a coherent whole. A software engineering process group (SEPG) facilitates software process definition and improvement efforts. An organization-wide training program is implemented to ensure that the staff and managers have the knowledge and skills required to carry out their tasks. Projects use the organization-wide standard software process to create their own defined software process that encompasses the unique characteristics of the project. Each project uses a peer review process to enhance product quality.

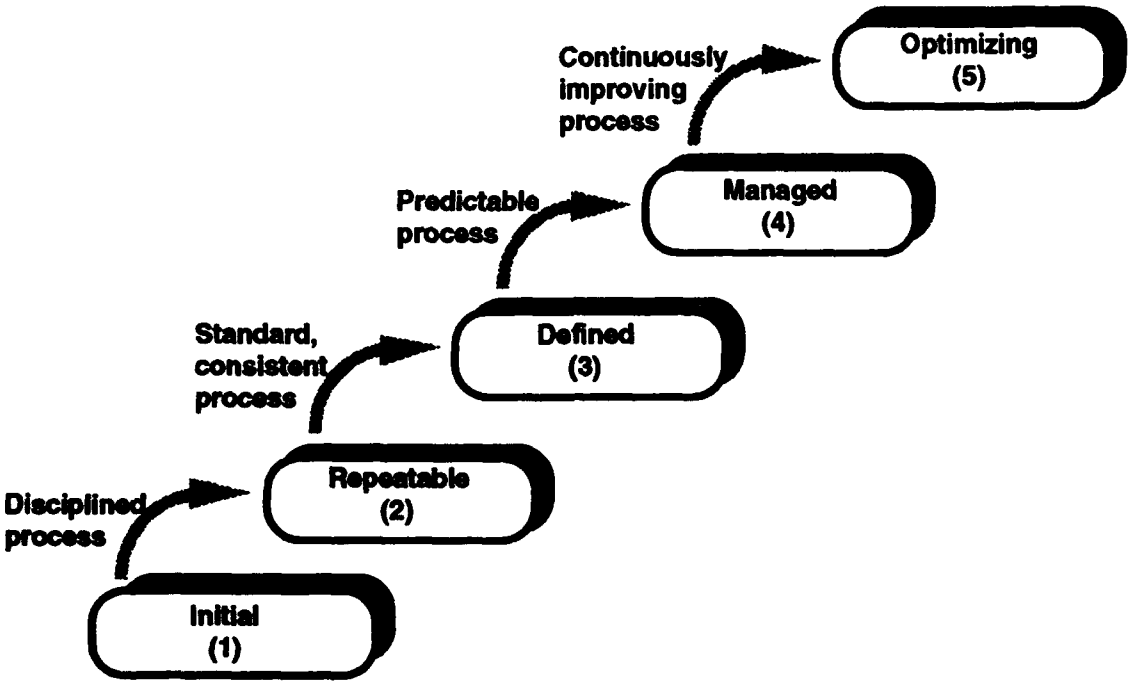


Figure 2.2-1. The Five Levels of Software Process Maturity

An organization at the Managed Level sets quantitative quality goals for software products. Productivity and quality are measured for important software process activities across all projects in the organization. A process database is used to collect and analyze the data from a carefully defined process. Software processes have been instrumented with well-defined and consistent measures that establish the quantitative foundation for evaluating project processes and products.

An organization at the Optimizing Level focuses on continuous process improvement. The organization has the means to identify weak process elements and strengthen them, with the goal of preventing the occurrence of defects. Statistical evidence is available on process effectiveness and is used in performing cost-benefit analyses on new technologies. Innovations that exploit the best software engineering practices are identified.

A maturity level is composed of several key process areas. Figure 2.2-2 displays the key process areas for each maturity level. The key process areas identify the issues that must be addressed to achieve a maturity level. In essence, they may be considered the requirements for achieving a maturity level. Each key process area represents a cluster of related activities that, when performed collectively, achieve a set of goals considered important for enhancing process capability. When these goals have been accomplished on a continuing basis, the organization can be said to have institutionalized the process capability characterized by the key process area.

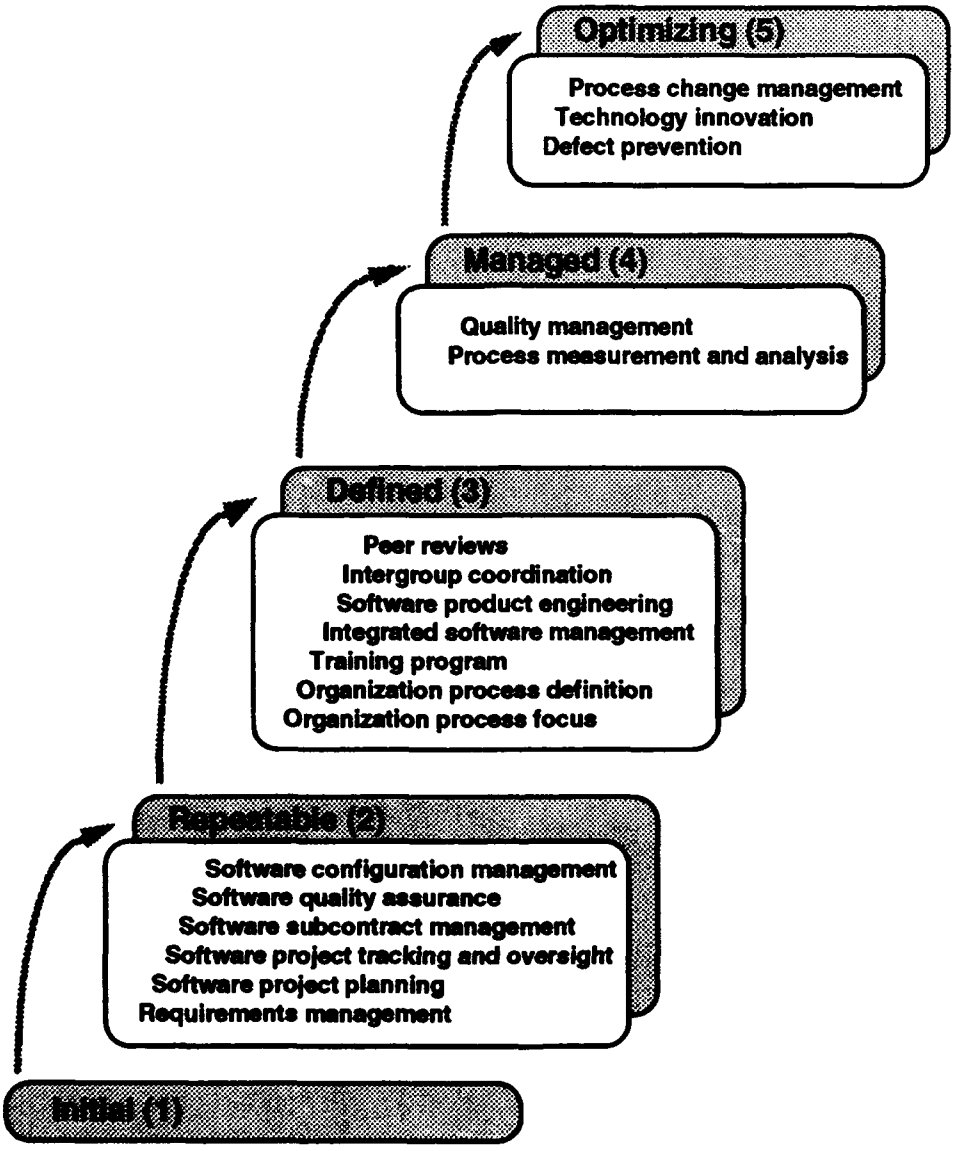


Figure 2.2-2. The Key Process Areas by Maturity Level

The CMM provides a wealth of information for process capability and process improvement. It also serves as the foundation upon which a measurement program can be built.

2.3. Goal-Question-Measure Paradigm

Basili and Weiss proposed a methodology for collecting valid software engineering data that has come to be known as the goal-question-measure paradigm [Basili 84]. The paradigm states that an organization should have specific goals in mind before data are collected. This ensures that only appropriate data are collected.

The first step for an organization is to establish the goals of its measurement program. The organization determines what it is that it is attempting to do with these data. Once the goals have been established, the organization in the second step develops a list of questions that are to be answered by the measurement program. Questions force the organization to define more sharply the goals of the measurement program. The questions ensure that only data appropriate to achieving those goals are collected. If a data item is not needed in analysis, it is discarded. Also if there is a goal for which questions cannot be formulated or for which data cannot be collected, then it is not a viable measurement goal and is discarded.

The final step is to collect the data. However, when these measures are now made, they are made for a well-defined purpose.

This paradigm was used in this task to maintain a focus on the measures appropriate to achieve the goals of the key process areas. There are many measures available for use in the software industry. Only those indicators that would help an organization in assessing and improving itself as well as providing useful information to the software project manager during software development were selected.

2.4. Selection Criteria for Indicators

The following criteria were used in selecting indicators:

- The indicator had to apply to a goal of a key process area within the CMM.
- The indicator could be used both as a status indicator and as a predictor of future status.
- The indicator is used in measuring software process improvement efforts.
- The indicator is an existing measure; that is, there is a reference to it in the literature.
- The indicator is easy to derive; that is, it does not require complicated arithmetic procedures.
- The input items to an indicator require straightforward data collection efforts; that is, the data collection lends itself to automation.
- The graphing of the trend charts is simple enough to be done by hand or lends itself to automation.

An indicator had to satisfy a majority of the criteria. Not every criterion had to be satisfied. Each candidate indicator was evaluated against these criteria and was

selected on how well it satisfied these criteria. The one criterion that every indicator had to satisfy is the first one: the indicator had to apply to a goal of a CMM key process area.

2.5. Assumptions

The following assumptions were made during this task:

- The key process areas are not likely to change in the first revision of the CMM. The CMM was released for comment in August 1991 with the understanding that it would be revised within a year. It is expected that some of the details in the key practices will change but that the general structure of the CMM (i.e., the specific key process areas) will remain the same.
- Projects within an organization may be at varying levels of process maturity. The indicators are applicable at the different maturity levels. Indicators can be used by both low and high maturity level projects. As a project and the organization as a whole mature, the indicators can be used to provide different levels of information. For example, at the Repeatable Level, a project would use the indicators primarily to determine its status. As the project and the organization mature, the indicators would be used for their predictive as well as current status capability.
- A lower maturity level project does not have to wait to achieve a higher maturity level to use an indicator discussed at the higher level. For example, if a project at the Repeatable Level wishes to use the indicators associated with the Peer Review key process area, it can do so, but the data may be less accurate, and the indicator may not be as effective.

2.6. Approach

The approach to this research began with two parallel activities: an analysis of the CMM for measurement activities and a literature search for references to software measures or indicators.

In the analysis of the CMM, a measurement reference, whether explicitly stated or inferred from the CMM text, was recorded wherever it was found within the key process areas. As expected, the majority of the measurement references are found in Activities Performed, Monitoring Implementation, and Verifying Implementation. Commitment to Perform and Ability to Perform provided information on who uses the measures.

The literature search quickly revealed the scope and diversity of software measures in the industry as well as the difficulty of obtaining detailed information on measurement programs within organizations. The search became quickly constrained to literature in the public domain as many organizations are protective of their measurement programs.

The literature search also showed that indicators, rather than basic measures, are more appropriate for this document. Indicators allow greater latitude on how to actually measure an item. For example, several ways of determining software size exist. These include counting lines of code, pages of documentation, or function points. By using an indicator of size, discussion can focus on trends in size changes and their implications. This discussion is independent of how size is measured. Whatever measure an organization uses to measure size, the discussion on variation in software size is still applicable.

The next step was to integrate the results of the CMM analysis and the literature search. This was accomplished by applying the goal-question-measure paradigm to the goals of the key process areas. Questions from the viewpoint of the project software manager and the software engineering process group were considered. Using these questions, the indicators contained in this document were selected.

3. Overview of the Indicators

The indicators chosen are grouped into the thirteen categories given in Table 3-1. Not all categories occur at all maturity levels. Table 3-2 and Table 3-3 provide a mapping of the indicators to the maturity level.

Appendix C explains the links between the indicator categories and the software measurement references of the Capability Maturity Model key practices. Appendix D maps the software indicators to software measurement references in the CMM key practices.

A careful reading of Appendix C and Appendix D shows that there should be risk indicators. However, no risk indicator is discussed in this document since the literature search and discussions with Software Engineering Institute personnel in the Risk Project revealed that the state of the practice in risk management is still too immature. However, the project software manager can use many of the indicators discussed in this document as risk analysis tools to determine project risks.

Indicator Category	Description
Progress	Provides information on how well the project is performing with respect to its schedule commitments.
Effort	Provides visibility into the contribution that staffing has on project costs, schedule adherence, and product quality.
Cost	Provides tracking of actual costs against estimated costs and predicts future project costs.
Quality	
Software Quality Assurance Audit Results	Provides estimation of product quality and compliance of staff to project processes.
Review Results	Provides status of action items from life-cycle reviews.
Trouble Reports	Provides insight into the quality of the product and processes and the effectiveness of testing.
Peer Review Results	Provides insight into the quality of the intermediate and final products and into the peer review and development processes.
Defect Prevention	Provides insight into the cause of defects and the effect of defect prevention activities on defect insertion rates.
Stability	
Requirements Stability	Provides visibility into the magnitude and impact of requirements changes.
Size Stability	Provides insight into the completeness and stability of the requirements and into the capability of the staff to complete the project within the current budget and schedule.
Process Stability	Provides insight into the effectiveness and quality of the defined process.
Computer Resource Utilization	Provides information on how well the project is performing with respect to its computer resource utilization goals/requirements.
Training	Provides visibility into the effectiveness of the organization's training program in meeting the skill requirements.

Table 3-1. Indicator Categories and Their Description

Indicator Category	Repeatable Level	Defined Level
Progress	Actual vs. planned completions Gantt chart	Actual vs. planned completions with ranges Gantt chart PERT chart
Effort	Actual vs. planned staffing profiles	Actual vs. planned staffing profiles with finer granularity
Cost	Actual vs. planned costs Cost and schedule variances	Actual vs. planned costs with ranges Cost and schedule performance indices
Software Quality Assurance Audit Results	Status of noncompliance issues	Status of noncompliance issues Audit information Sampling size information
Review Results	Status of action items	Status of action items
Trouble Reports	Status of trouble reports Number of trouble reports opened, closed, unevaluated during reporting period Trouble report density Comparison of trouble reports and test cases passed	Status of trouble reports Number of trouble reports compared with historical data Length of time trouble reports remain open Number of trouble reports per product
Peer Review Results		Number of opened and closed defects Number of defects by product defect density Pareto analysis of defects Preliminary control charts
Defect Prevention		
Requirements Stability	Number of requirements changes and requirements clarifications Distribution of requirements over releases	Number of requirements changes and requirements clarifications with ranges Distribution of requirements over releases Distribution of requirements changes by requirement type Length of time requirements change requests remain open Number of waivers requested and approved from requirements
Size Stability	Size growth Distribution of size over releases	Size growth with ranges Distribution of size over releases
Process Stability		Number of process changes Number of waivers from process
Computer Resource Utilization	Actual vs. planned profiles of computer resource utilization	Actual vs. planned profiles of computer resource utilization
Training		Actual vs. planned number of classes offered Actual vs. planned attendance Course quality profiles Number of waivers from training

Table 3-2. Indicators for the Repeatable and Defined Levels

Indicator Category	Managed Level	Optimizing Level
Progress	Actual vs. planned completions with control limits Gantt chart PERT chart	Ratio of rework time to total project time per project Rate of time spent in activities undergoing process change
Effort	Same as Defined Level	Ratio of rework effort to total project effort per project Rate of effort spent in activities undergoing process change
Cost	Actual vs. planned costs with control limits Cost and schedule performance indices	Comparative costs and benefits of alternative process improvement and defect prevention activities and technologies Actual vs. planned cost and benefit of an alternative process improvement activity, defect prevention activity, or technology
Software Quality Assurance Audit Results	Same as Defined Level	Same as Defined Level
Review Results	Same as Defined Level	Same as Defined Level
Trouble Reports	Causes of trouble reports Testing, development, and implementation efficiency	Same as Defined and Managed Levels
Peer Review Results	Number of open and closed defects Number of defects by product defect density Pareto analysis of defects Control charts on peer review characteristics	Same as Defined and Managed Levels
Defect Prevention		Defect category profiles Defect insertion rates
Requirements Stability	Same as Defined Level	Same as Defined Level
Size Stability	Size growth with control limits Distribution of size over releases	Same as Defined and Managed Levels
Process Stability	Same as Defined Level	Same as Defined Level
Computer Resource Utilization	Same as Defined Level	Same as Defined Level
Training	Same as Defined Level	Same as Defined Level

Table 3-3. Indicators for the Managed and Optimizing Levels

4. The Repeatable Level—Maturity Level 2

This chapter summarizes the characteristics of an organization with a repeatable process and discusses the indicators that are appropriate for the Repeatable Level.

4.1. Characteristics of a Repeatable-Level Organization

An organization with a repeatable process is characterized as one with basic project management processes in place that allow the organization to develop a project plan and to track software costs, schedule, and functionality against this plan. Results from previous projects allow realistic commitments to be made for the current project. Infrastructures are in place to allow proper software quality assurance, configuration management, requirements management, and subcontract management on the project. Mechanisms are in place that allow the project to make changes to the plan when appropriate.

Indicators appropriate for a Repeatable-Level organization are progress, effort, cost, quality, stability, and computer resource utilization.

4.2. Progress

Progress is measured through the completion of activities on the project schedule. Progress indicators are used to monitor progress in terms of task completion and task output. The difference between the actual and planned completions is an indication of project adherence to the plan. Significant deviations (as determined by the project or organization) indicate problems. Furthermore, the progress indicators can show trends that suggest potential future problems.

Progress indicators may be used to monitor activities throughout all life-cycle stages. Each stage has activities that must be completed and that can be quantitatively measured with process measurements. Each activity also produces tangible outputs such as source code or completed design reviews whose effectiveness can be monitored with product measurements. These measures and others provide managers with information not only about the product, but also about the processes followed to develop the product.

Progress indicators may be used by all levels of project management. The first-line managers prepare their schedule of activities, which includes planned start and completion dates, and record the actual dates these activities occur. The next level of managers receive this information, form a composite schedule for their area of control, and monitor progress against the planned schedule. Schedules and reports are consolidated so that the project manager obtains an overall view of the project progress, but the lower-level data are available, when necessary, to provide detailed information.

Objective of the Progress Indicators

To provide software managers with information on the progress of the project within each life-cycle development activity.

Indicators

- The deviation of the actual project progress from that planned
- The trend in the rate of progress

Key Process Area Goals Addressed

Software Project Planning:

- A plan is developed that appropriately and realistically covers the software activities and commitments.
- The software estimates and plans are documented for use in tracking the software activities and commitments.

Software Project Tracking and Oversight:

- Actual results and performance of the software project are tracked against documented and approved plans.

Software Subcontract Management:

- The prime contractor tracks the subcontractor's actual results and performance against the commitments.

Life-Cycle Stage: All

Users: All levels of project software management

Users' Questions

- Are the schedules of activities for the project consistent with each other and with the software requirements and the software development plan?
- Is the software subcontractor's planned schedule consistent with the software requirements and the software development plan?
- Are the actual results and performance of the software project tracked against the software development plan?
- Are the actual results and performance of the software subcontractor tracked against the software development plan?
- Does the actual performance of the project indicate schedule slippage and the necessity of a replanning effort?
- Does the actual performance of the subcontractor indicate schedule slippage and the necessity of a replanning effort?

Input

- Project software development plan
- The planned and actual start and completion dates for the software development activities
- A count of tasks or task outputs planned and actually produced in the current reporting period. Task outputs include:
  - Requirements analyzed
  - Software requirements documented
  - Test cases written
  - Design elements (e.g., design diagrams) completed
  - Units designed, coded, unit tested, and integrated
  - Documents completed
  - Test cases executed
- Planned software size and the size actually completed and recorded during each reporting period

Note that the planned start and completion dates are the estimated dates.

This is a recommended set of items to monitor from the Repeatable Level in the Capability Maturity Model for Software (CMM). Additional items can be monitored as long as they are planned or estimated and are measurable.

Interpretation

During project planning, software managers prepare Gantt charts from the list of major activities in the project software development plan. Lower-level managers determine the activities they are required to perform to satisfy the major project activities. They then prepare Gantt charts that show the start and completion dates of their activities.

When preparing a Gantt chart, the project manager has the responsibility of ensuring that the various software development activities carried out by the subcontractor and project personnel are consistent with each other, the software requirements, and the software development plan.

Figure 4.2-1 shows a typical Gantt chart. For each activity, the original baseline activity duration is plotted along a timeline. The planned and actual start and completion dates are marked on the bar with triangles. The start date is marked at the left end of the timeline and the end date at the right. The bar is also marked with the current baseline dates, if different from the original baseline. The actual start and completion dates are marked along the bar. Also, the amount of schedule slippage is recorded next to the bar. The schedule slippage indicates the amount of progress deviation. Gantt charts can also show the total number of activities completed and the number of activities delayed.

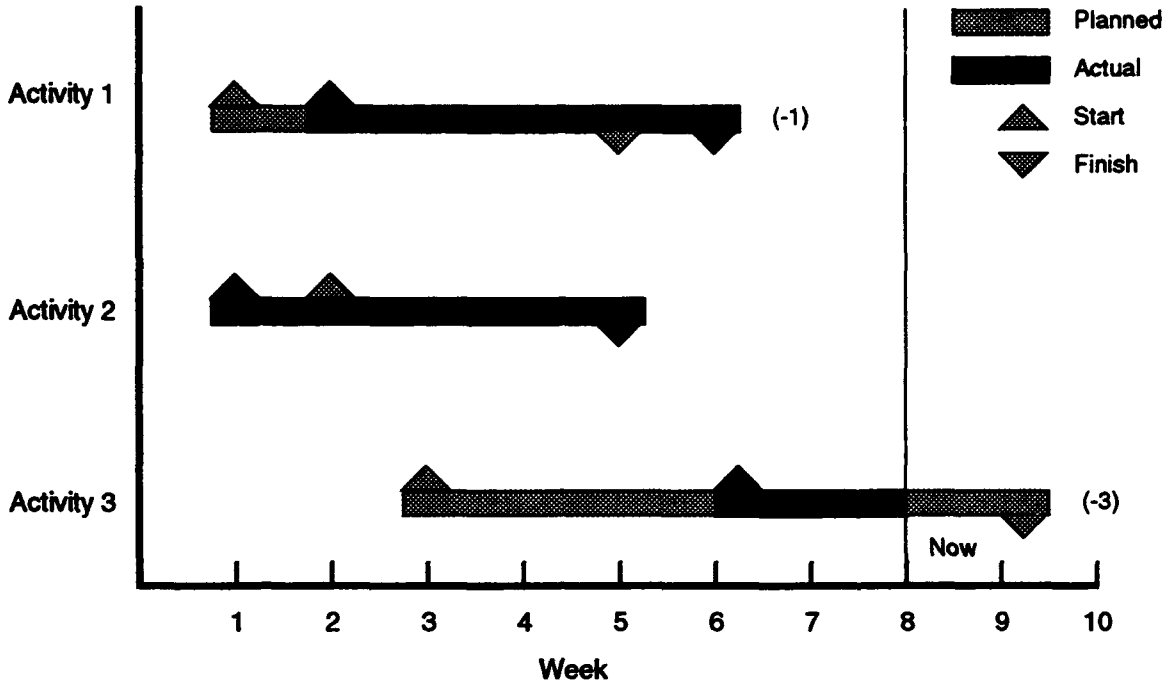


Figure 4.2-1. A Typical Gantt Chart

Figure 4.2-1 shows that activities 1 and 2 are completed and that work is still in progress for activity 3. Activity 1 started one week late and finished one week late. This indicates that project personnel maintained their planned productivity for this activity once it began. Activity 2 began one week early but finished as scheduled. The completion triangle is superimposed on the planned triangle. Activity 3 began three weeks late. The manager believes that this activity is progressing with its planned productivity since the projected schedule slippage at this time is only the three-week delay in the start date. The manager uses the planned versus actual charts for that activity to monitor the productivity of the staff.

Figure 4.2-1 has a Now timeline (i.e., a vertical slice through the chart) that indicates the date when the last measurement was made. An alternate approach is to use a schedule line of balance. This timeline marks where the project is on each activity line on the date of the report. It emphasizes which activities are ahead or behind schedule.

Information from a Gantt chart can be used to produce another chart, as shown in Figure 4.2-2, where the planned dates for two consecutive events are compared.

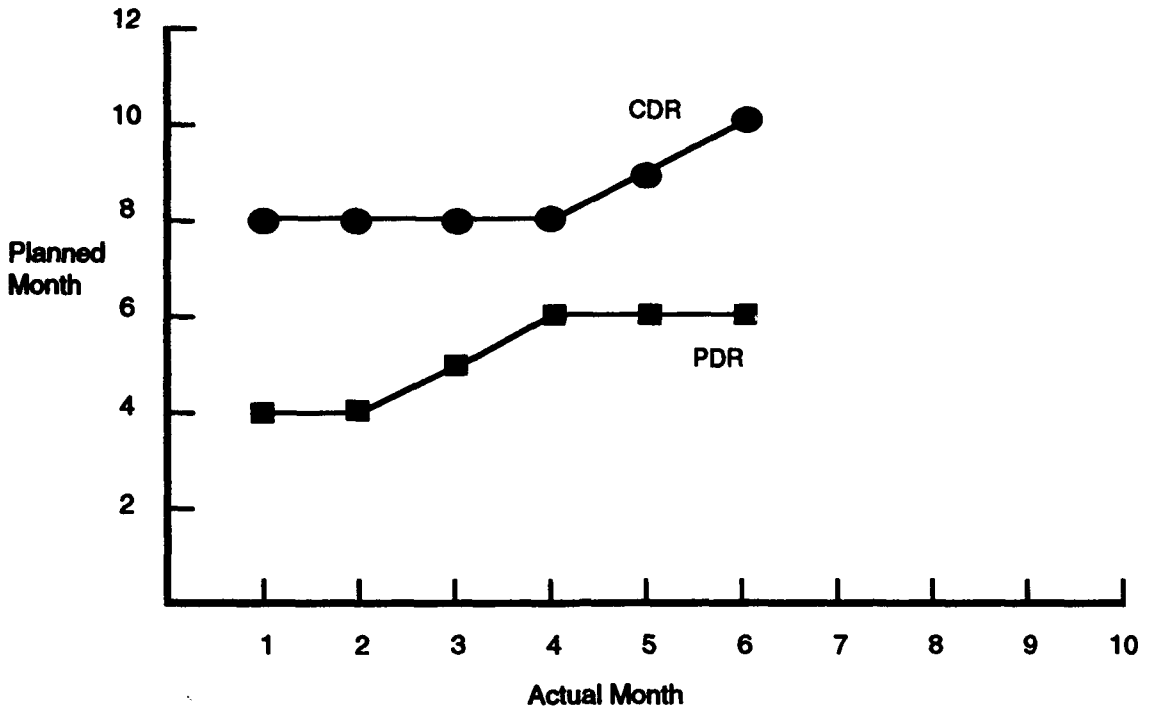


Figure 4.2-2. Comparison of Time Between Two Consecutive Events

In Figure 4.2-2, the planned dates of the preliminary design review (PDR) and the critical design review (CDR) are plotted against time. At month 1, the PDR was planned for month 4 and CDR for month 8. At month 3, however, the PDR has slipped to month 5 while the CDR is still scheduled for month 8. As the PDR continues to slip, the manager should be questioning why the CDR is not slipping. Finally at month 5, the schedule for the CDR begins to change. Based on the earlier schedule slippage of the PDR, the CDR could have been rescheduled earlier and more realistically to month 10.

Other progress indicators include the deviations and trends derived from graphs depicting the cumulative number of planned and actual completions for an item. For each item, the planned completions (based on current baseline) and the actual completions to date are plotted against time. (The project defines its completion criteria. For example, the project may decide to consider a unit design complete when the design has passed its peer review.) Figure 4.2-3 shows a typical actual-versus-planned plot. The Now timeline shows the date of the last measurement.

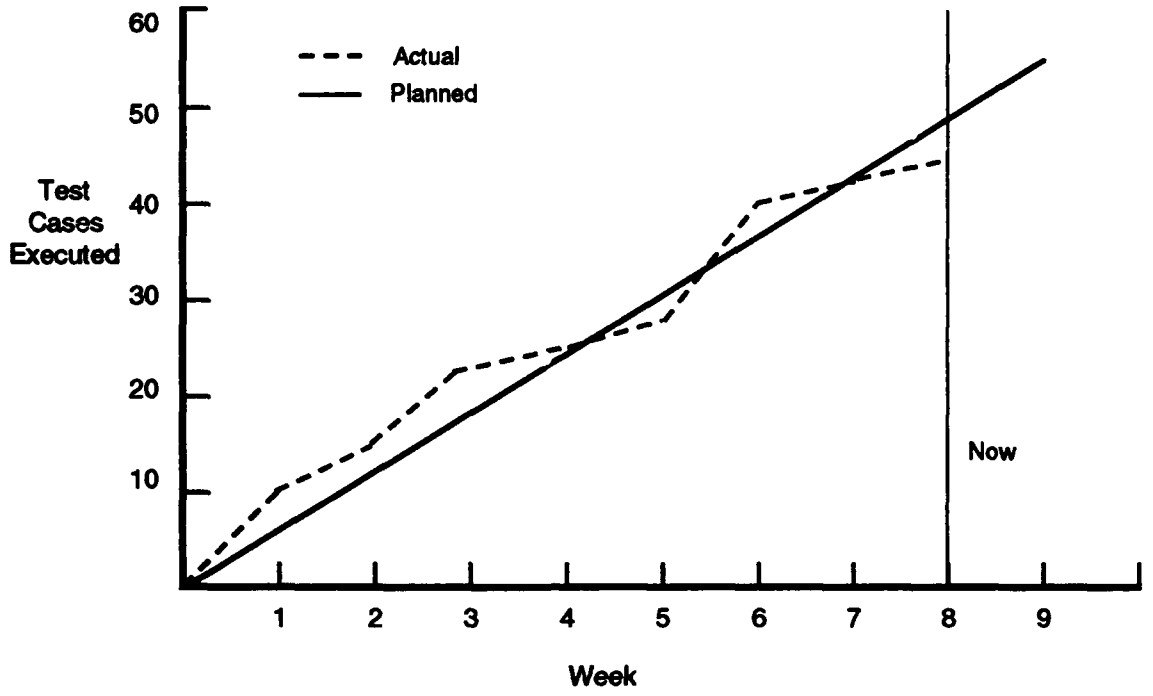


Figure 4.2-3. Actual Completions Compared with Planned Completions

In Figure 4.2-3, the cumulative actual-versus-planned completion of test cases executed are compared. The manager should expect some deviation from a constant rate to test completion. This occurs frequently when a critical number of problems has been identified by testing and must be resolved by the development group before further significant testing can continue. The actual number of tests executed and/or passed might drop below the planned number, but it should quickly reach and pass the planned number after the corrections have been made. Therefore, the actual numbers may oscillate around the planned number. The extent of the deviation between the planned and actual completions and the trend in the deviation indicate the readiness of the computer software configuration item and/or the system for testing, or they may indicate how much additional time is required to complete testing.

Separate graphs are prepared for the items listed in the input section depending on the life-cycle stage of the project. Size is given as one of the input items. It can also be used as a stability indicator and is discussed as such.

Because of replanning, the content, ordering, or timing of tasks and milestones may differ from the original baseline. Therefore, the number of task completions, the size, or other progress measures may be affected. Figure 4.2-4 shows how the original baseline was changed to the current planned baseline to accommodate the actual, rather than planned, productivity.

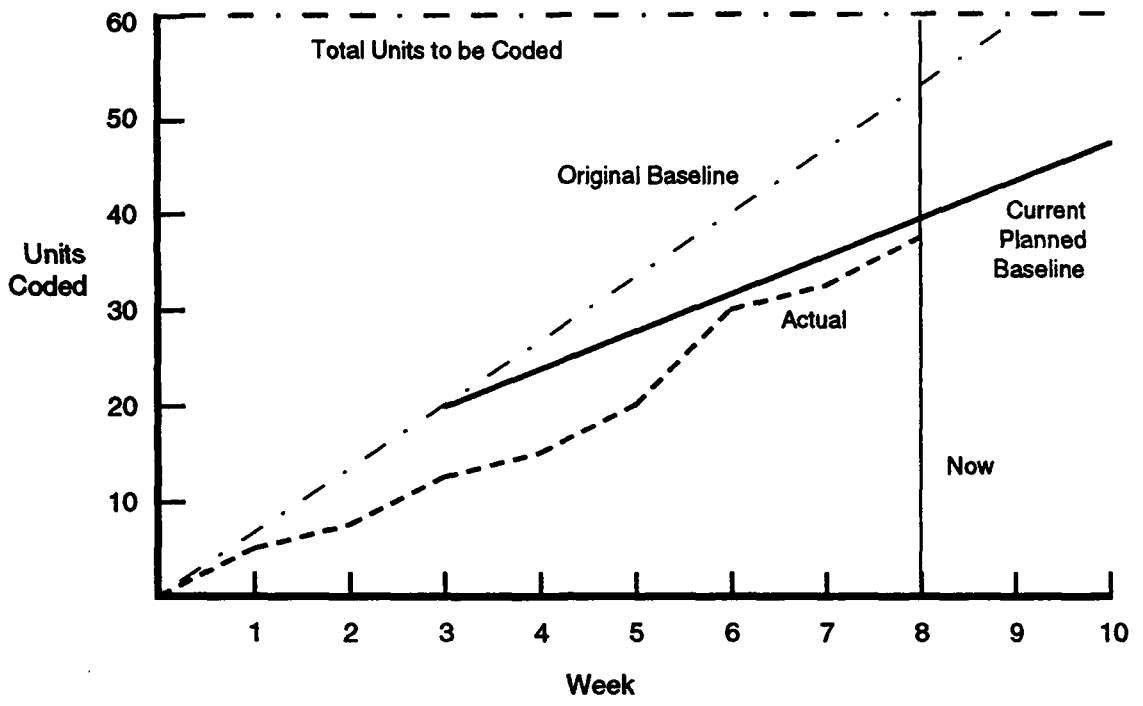


Figure 4.2-4. Change in Original Baseline

In Figure 4.2-4, the manager replanned the completion of the number of units coded at week 3. Since the total number of units to be coded has not changed, the manager has accepted the fact that more time is required to complete this activity. The manager calculates a slippage of approximately four weeks. This number is derived by determining where the current planned line intersects the total number of items to be completed and noting the difference between this time and the original completion time.

The original baseline may change due to a change in the number of items to be completed. Figure 4.2-5 shows how the planned baseline changed as the total number of software requirements grew.

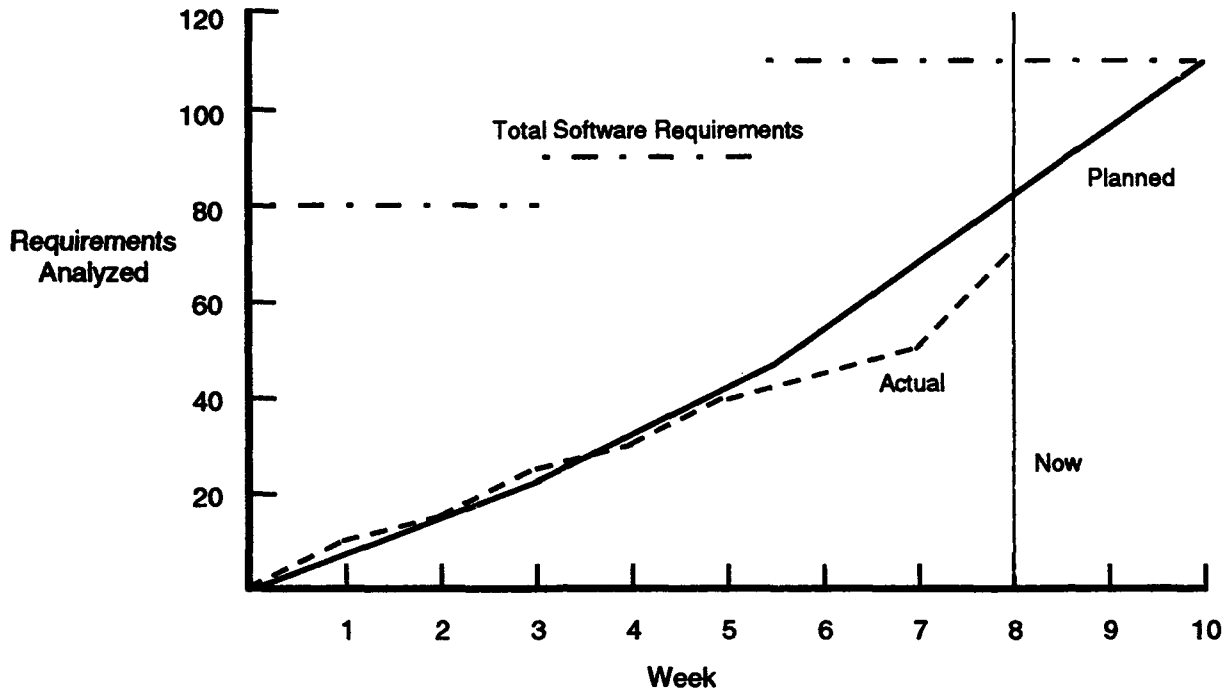


Figure 4.2-5. Actual and Planned Completions with Different Baselines

In Figure 4.2-5, all requirements are scheduled to be analyzed by week 10. At week 3, ten requirements were added. The planned line shows an increase in slope to account for the additional requirements. During week 5, twenty more requirements were added. Again, the slope of the planned line increases since all requirements are scheduled to be analyzed by the end of week 10. If the productivity of the analysts does not change (there is no reason why it should at this time), then Figure 4.2-5 implies that more staff have been added to complete the requirements analysis.

The baseline in Figure 4.2-5 is changed to provide a more accurate report on the progress during the current reporting period. If the original baseline had not changed to reflect the growth in software requirements, the deviation of the actual and planned completion could become unrealistically large.

The manager needs to determine the cause of any deviation in actual-versus-planned progress. For example, in Figure 4.2-5, is the fact that fewer requirements have been analyzed than planned the result of the experience level of the staff or the result of poor planning? Are the personnel less productive than anticipated?

Sources

[AFSC 86] discusses planned and actual completions graphs at the computer software configuration item (CSCI) level.

[Decker 91] lists requirements diagrams; function specifications; design diagrams; test cases; units designed, coded, and tested; modules tested; and computer software components tested as items tracked on planned and actual completions graphs.

[Grady 87] states that calendar measures are part of the Hewlett-Packard metrics program.

[Landis 90] discusses planned and actual completions graphs for units coded, read, and tested.

[Rozum 92] has a discussion on Gantt charts in their milestone performance metric and a discussion of planned and actuals in their development progress metric.

[Schultz 88] discusses software requirements documented in his design progress metric and the number of computer software units (CSU) designed, coded, tested, and integrated in his CSU development progress metric. He also discusses the planned and actual completions of CSCIs integrated in his test progress metric.

[STEP 91] discusses a schedule metric upon which Figure 4.2-2 is based and a development progress metric.

4.3. Effort

Determining the number of staff needed at any one time is an important function performed by the project software manager. This planning begins once the manager has determined which work items need to be performed in developing the software products. A work breakdown structure is useful in defining and placing the individual work items into manageable work packages. One or more work packages form a cost account. Staff with the appropriate skills are assigned to perform the work contained in the work packages, according to the project schedule. By summing the number of staff during each reporting period, the project software manager establishes the staffing profile for the project.

The effort indicators allow the project software manager to track the actual and planned expenditure of personnel resources. In addition, the manager uses them to ensure that the appropriate skills are being applied to each work package and to indicate when different skill categories are needed during the project. If the project is following its staffing profile, the manager may assume that the risk to the project from staffing issues is low. If a manager is unable to obtain staff with the appropriate skills, the manager can use the staffing profile to determine when these skills are necessary and plan accordingly for the appropriate training. In addition, the manager can use the profiles to determine the amount of office space required to house the staff. This is particularly important when a facility is running near capacity. The manager who knows when an increase in the number of staff is to occur can take the necessary steps to ensure that facilities are available when the additional staff arrives.

Effort indicators may be used by all levels of project management to compare the actual profile against the plan. During project planning, the first-line managers prepared their planned staffing profiles. During the implementation of the plan, they compare the actual with the planned profiles. Each level of management receives this information from the lower-level managers, forms a composite profile for the area of control, and monitors the actual profile against that planned. Reports are consolidated so that the project manager obtains an overall view of the project profile, but the lower-level data are available, when necessary, to provide detailed information.

Objective of Effort Indicators

To provide software managers visibility into the contribution that staffing has on project costs, schedule adherence, and product quality and the amount of effort required for each process.

Indicators

- Trends in the actual staffing levels
- Staffing profile by labor category
- Profile of unplanned staff losses

Key Process Area Goals Addressed

Software Project Planning:

- A plan is developed that appropriately and realistically covers the software activities and commitments.
- The software estimates and plans are documented for use in tracking the software activities and commitments.

Software Project Tracking and Oversight:

- Actual results and performance of the software project are tracked against documented and approved plans.
- Corrective actions are taken when the actual results and performance of the software project deviate significantly from the plans.

Software Subcontract Management:

- The prime contractor tracks the subcontractor's actual results and performance against the commitments.

Life-Cycle Stage: All

Users: All levels of project software management

Users' Questions

- Has the software planning process resulted in a staffing profile that is consistent with the planned schedule and budget?
- Is the software subcontractor's planned effort consistent with its planned schedule and budget?
- Is enough effort being planned and applied to the project to achieve the desired schedule?
- Does the actual expenditure of effort indicate that a replanning effort is necessary?
- Is the rate at which effort is being expended going to overrun/underrun the planned effort?
- Will the types of effort being applied to the contract have an impact on the quality of the final product?

Input

- Planned total staff-hours by time period
- Actual total staff-hours expended during time period
- Total number of staff
- Planned staff-hours by time period for each cost account element
- Actual staff-hours expended during the time period for each cost account element
- Planned staff-hours by life-cycle stage

- Actual staff-hours expended during life-cycle stage
- Planned staff-hours by time period by labor category
- Actual staff-hours expended during time period by labor category
- Number of staff by labor category
- Number of unplanned staff losses during time period

Interpretation

Figure 4.3-1 compares the actual total number of staff with the planned number for each month. The time period used for the reporting period is determined by the project. One contributing factor to the reporting frequency is the ease of obtaining the actual staff-hours expended from the project control office.

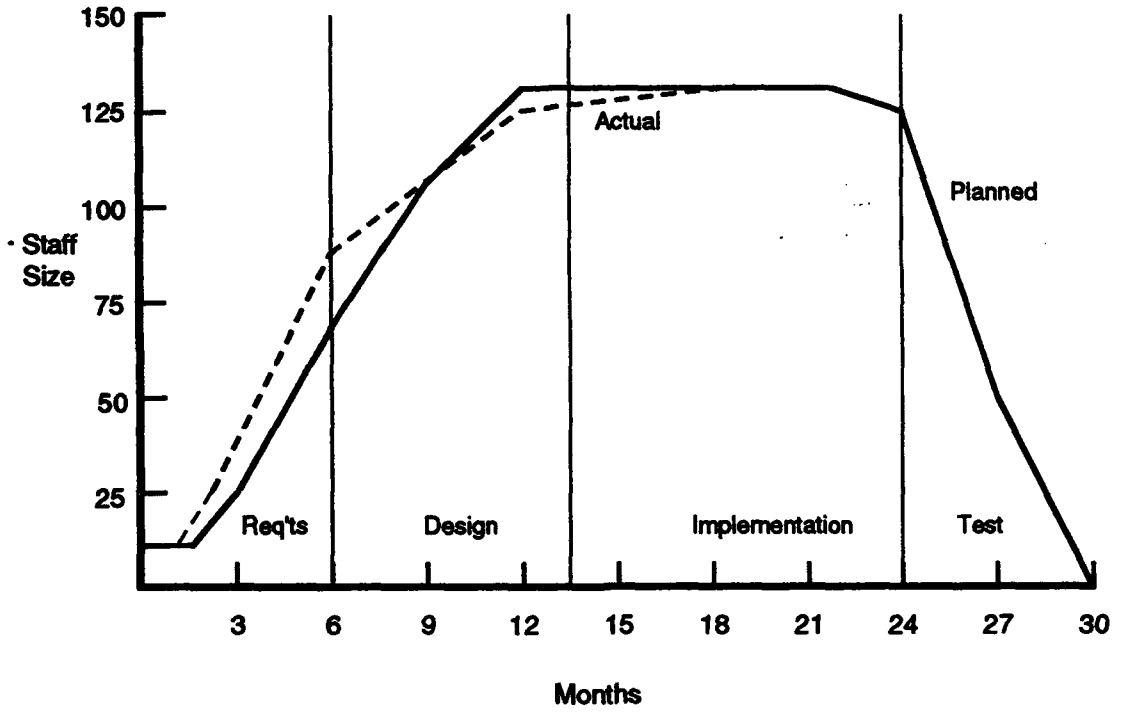


Figure 4.3-1. Total Planned and Actual Staffing Profile

The planned profile in Figure 4.3-1 is typical for software development projects. The figure shows the following:

- An increase of staff occurs until implementation. This increase must be reasonable since time is required for the hiring process.
- Peak staffing occurs during implementation since coding is labor intensive.

- Staffing remains relatively stable during implementation.
- Staffing declines rapidly during testing since a full complement of implementors is not required to correct problems detected during testing. When it is necessary to retain a large number of implementors due to a large number of trouble reports written during testing, the manager should question the quality of the product.

In general, if the staffing profile is too pointed, communication/coordination problems may occur [Brooks 82]; if the profile is too flat, the schedule could be shortened.

Figure 4.3-1 shows the number of staff for the entire project. Each lower-level manager prepares similar graphs for the cost accounts under their control. The numbers are consolidated for each level of management until the final graph shows the staffing profile for the entire project. As part of the planning process, the lower-level managers and the project software manager check their staffing profiles with the progress and cost indicators to ensure that the three indicators form a consistent plan before work begins.

Managers should expect deviation of the actual number of staff from the planned number during the reporting period. However, large deviations or an extended overstaffing or understaffing period requires analysis. An understaffing situation, extending over several contiguous reporting periods, may be the result of the following:

- An overestimate of the software size
- Insufficient progress
- An increasing number of open trouble reports
- Lack of understanding of requirements
- A very productive team
- A poor quality product

The manager uses other indicators, especially progress, cost, and quality, to determine the correct explanation. If the true reason is that productivity is not being maintained, then the understaffing is an early indication of schedule slippage. The manager still has time to take corrective action before the schedule slippage becomes severe. Brooks has pointed out that adding staff to a late project seldom improves the schedule and often causes additional delay [Brooks 82].

An overstaffing situation, extending over several contiguous reporting periods, may be the result of the following:

- A more complex problem than expected
- Unstable requirements, which cause extensive rework
- Staff with the inappropriate skill for the work being performed
- Insufficient progress
- Increasing number of trouble reports
- An underestimate of the software size

As in the case of the understaffing situation, the manager can use other indicators to determine the reason for the overstaffing.

Once the planned staffing profile has been developed, the project software manager can determine from the organization's historical data whether the allocation of staff over the life-cycle stages is appropriate for that type of software project. Life-cycle stages are shown in Figure 4.3-1. The manager can derive the percentage of staff-hours planned for each life-cycle stage and compare them to historical data for similar projects. If there are significant deviations, the manager needs to decide whether the allocation is incorrect or whether the project has some unique characteristics that make it different. One example of how the staffing profiles can be used is reported by Landis et al [Landis 90]. For their type of projects at the Goddard Space Flight Center, they find that the initial staffing level should be at least twenty-five percent of the average staffing level.

Figure 4.3-1 may be used in conjunction with progress indicators to give the manager an indication of the status of the project. For example, consider the four cases:

1. Actual staff-hours expended is greater than planned, and actual progress is greater than planned. Even though more effort is being spent, if the trend continues, the project will be completed ahead of schedule and within the planned amount of effort.
2. Actual staff-hours expended is less than planned, but the actual progress is greater than planned. Here less effort is required to complete the work. If the trend continues, the project will be completed ahead of schedule and with less effort than planned.
3. Actual staff-hours expended is greater than planned, and the actual progress is less than planned. In this case, more effort is required to complete less work. If the trend continues, the project will be completed later than planned and exceed the planned amount of effort.
4. Actual staff-hours expended is less than planned, and the actual progress is less than planned. This is a typical understaffing situation. If the trend continues, the project will finish later than scheduled.

Figure 4.3-1 shows the staffing profile for the entire project. A plot of the staffing profile by labor category or experience level, as in Figure 4.3-2, is also useful.

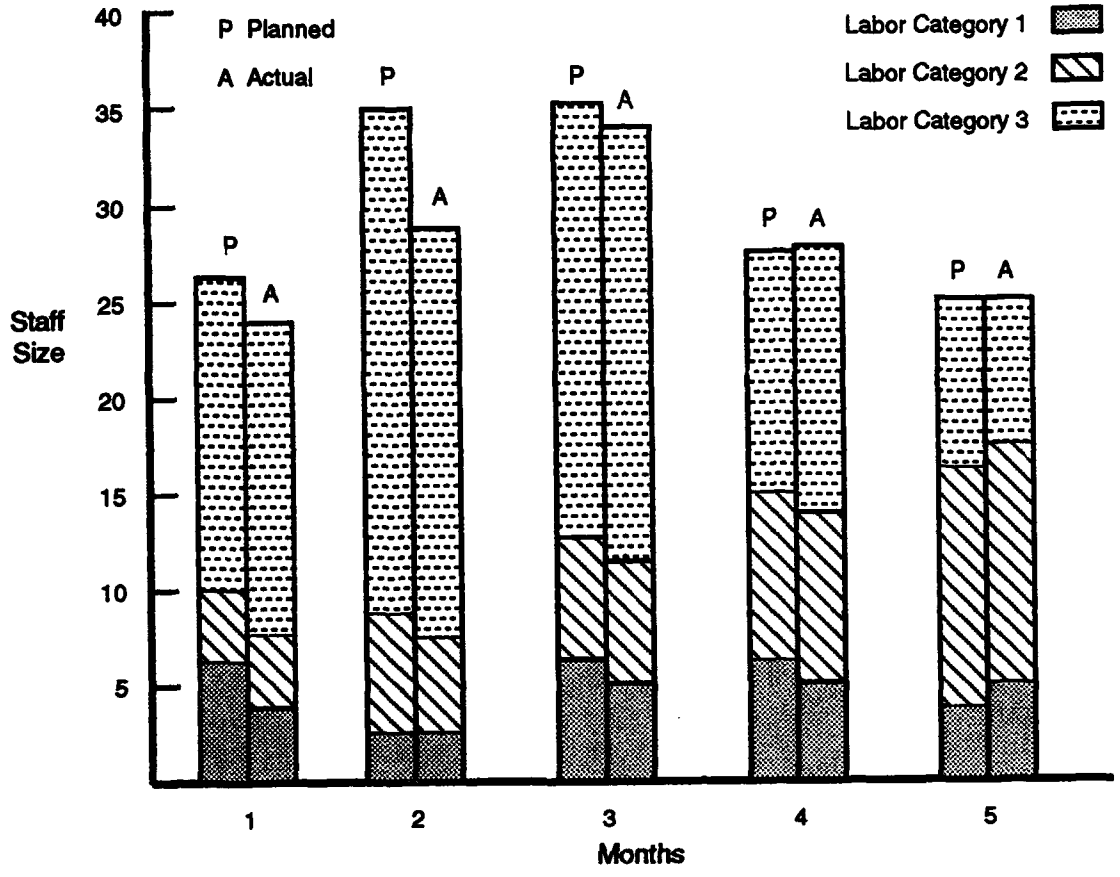


Figure 4.3-2. Staffing Profile by Labor Category

The manager of the project in Figure 4.3-2 can use the information in the figure to determine when it is necessary to add staff for a specific labor category. The manager needs to have not only the right number of people, but also the right people for the project. The figure also indicates when to release the appropriate personnel from the project or when project personnel would be free to go to other projects within the company. Typically, the more experienced people are needed early in the project during requirements and design. The greatest number of staff is needed during implementation. Testing usually requires fewer people.

Figure 4.3-3 shows the total number of staff, the number of staff added, and the number of staff lost for one labor category. Such a figure can be drawn for each labor category. The figure can also be drawn for staff experience (average years experience/person) in place of the labor category.

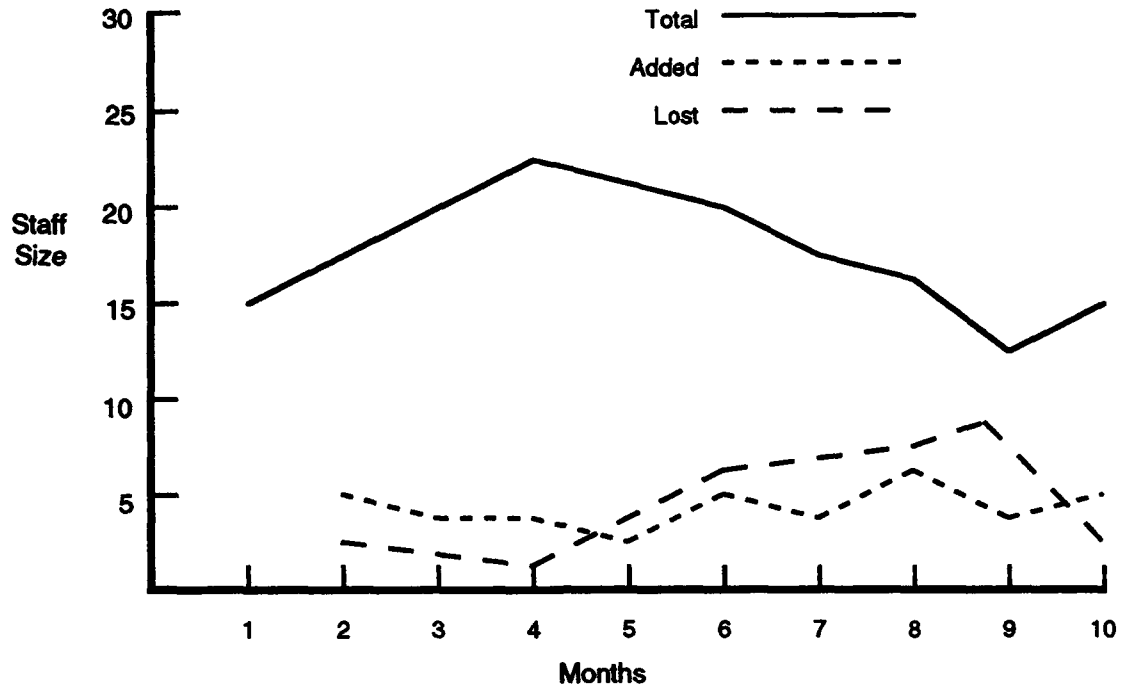


Figure 4.3-3. Typical Staffing Profile Including Staff Addition and Losses

The losses of most concern are those that are unplanned. For example, if the losses are attributable to a planned down staffing, there is no cause for concern. If, on the other hand, there is a number of requirements analysts who suddenly leave the project at the beginning of or during requirements analysis, the manager has cause for concern since their departures were not planned. Furthermore, the manager wants to know whether the personnel who left unexpectedly are being replaced by other personnel with similar skills. When drawn for the desired skill mix, Figure 4.3-3 can help answer that question.

Turnover rate can be monitored with this figure. Turnover is defined as the number of unplanned staff losses during the reporting period. Since these losses are unplanned, the personnel need to be replaced. There will always be some turnover on the project, but if the turnover rate becomes high (typically more than ten to fifteen percent), then the manager needs to determine the reason for the turnover. For example, the morale may be low on the project. A project wants to maintain a low turnover rate to retain continuity within the project, especially between requirements and design activities and design and implementation activities.

The figures used in the preceding discussion illustrate the use for the project as a whole. They may also be drawn for each computer software configuration item (CSCI), or if the software is developed in builds, they can be drawn for each build. The project is responsible for deciding the granularity of the figures, both with respect to the reporting period and the content.

Sources

[AFSC 86] discusses planned and actual staffing total profiles and staffing losses in its software development personnel indicator.

[Decker 91], [Landis 90], and [Pfleeger 89] discuss the use of planned and actual staffing profiles.

[Grady 87] reports that staff issues are part of the Hewlett-Packard software metrics program.

[IEEE 1045] discusses the experience level, size, and turnover rates of the project staff.

[Rozum 92] discusses planned and actual staffing total profiles, experience profiles, and also planned and actual staffing losses in their effort and staffing metrics.

[Schultz 88] discusses planned and actual staffing total profiles, experience profiles, and also planned and actual staffing losses in his software personnel metric.

4.4. Cost

Estimating cost is an important activity for the success of a project. Managers must define the work in their area, determine the skill level required to perform the work, and use productivity estimates and schedule constraints to determine budgeted costs over time. Most organizations working on federal government contracts use a work breakdown structure (WBS) to define work packages and often use the WBS to plan and track costs. For tracking, the managers track actual costs against the budget for their respective areas of responsibility. For reporting, costs are consolidated, that is, lower-level software managers report their costs to their manager who reviews and analyzes each of the reports and then prepares a report for the next level of manager. The project software manager receives a consolidated report for all software on the project. For example, the project software manager may receive a report for each computer software configuration item (CSCI) and the CSCIs combined. Since the lower-level reports are available, the project software manager can elect to review those with known or potential problem areas or those with high risk areas.

Objective of the Cost Indicators

To track actual costs against the project plan and to predict future project costs.

Indicators

- Trend in the actual cost of work performed (ACWP)
- Trend in the budgeted cost for work performed (BCWP)
- Trend in the cost variance. Cost variance (CV) is the difference between the budgeted cost for work performed and the actual cost of the work performed (CV=BCWP-ACWP).
- Trend in the schedule variance. Schedule variance (SV) is the difference between the budgeted cost for the work performed and the budgeted cost for the work scheduled (SV=BCWP-BCWS).
- Estimated cost at completion. The sum of the actual costs to date and the estimated costs of the work remaining.

Key Process Area Goals Addressed

Software Project Planning:

- A plan is developed that appropriately and realistically covers the software activities and commitments.
- The software estimates and plans are documented for use in tracking the software activities and commitments.

Software Project Tracking and Oversight:

- Actual results and performance of the software project are tracked against documented and approved plans.

Software Subcontract Management:

- The prime contractor tracks the subcontractor's actual results and performance against the commitments.

Life-cycle Stages: All

Users

All levels of management. The managers cited here are not restricted to software development managers, but include managers of support groups, for example, software quality assurance, software configuration management, training, etc.

Users' Questions

- Are the actual results and performance of the software project following the plan?
- Is the software subcontractor's planned budget consistent with the effort and schedule required to develop that portion of the software and with the project software development plan?
- Is the planned budget revised according to established procedures when the actual results and performance data indicate a replanning is necessary?
- Has the software planning process resulted in a software development plan in which the planned budget is consistent with the effort and schedule required to develop the software and the software requirements?

Input

- Budgeted cost for work scheduled (BCWS): the sum of the planned costs for each work package that has work scheduled to be accomplished during the reporting period.
- Budgeted cost for work performed (BCWP): the sum of the planned costs for each work package that has work accomplished during the reporting period. Work may be accomplished on work packages that have not been scheduled.
- Actual cost of work performed (ACWP): the sum of the costs incurred for each work package that has work accomplished during the reporting period.
- Budgeted cost at completion: the total budgeted cost of the work scheduled on the project.

These items are often tied to an "earned value" system in which the activity must pass some criterion before the work package is said to be completed.

Interpretation

The main feature of the cost indicator at the Repeatable Level is that it provides a comparison of the actual costs and the budgeted costs over time. The cost variance, schedule variance, and estimated cost at completion are obtained from a simple plot of the actual-versus-planned costs. Deriving a budget that is consistent with effort, schedule, and the project requirements is the crux of project planning. Simply stated, project planning involves the following steps:

- Determine the work to be performed
- Compute the effort required to perform the work
- Develop the schedule for performing the work
- Adjust the above estimates for project constraints
- Convert effort to cost
- Assess feasibility of cost and schedule

One approach for determining the work that must be performed is to use a WBS. The underlying philosophy of a WBS is to take a large task, for example, the development of a software system, and break it down into smaller pieces that are more manageable. These pieces can be used to estimate and track costs.

Once the work has been decomposed into manageable work packages, the manager determines the skills required to perform the work and the number of staff of each skill level required, and then schedules when the work will begin and end for each work package. The manager makes adjustments to the effort and schedule based on project constraints, schedule requirements, and the current software size estimate.

The manager next determines the cost associated with the work. Using the appropriate labor rates, the manager determines the personnel cost associated with each work package and adds any non-personnel costs to determine the cost associated with the work accomplished on the work package per reporting period. In the examples in this section, the reporting period is monthly. The project software manager then determines whether the total software costs are within the overall budget allocated and assesses the feasibility of the cost, effort, and schedule estimates. The manager iterates through this process until the final cost, effort, and schedule form a consistent package that is achievable.

Even though the project software manager has the ultimate responsibility for the planning, tracking, and controlling of the software project, every manager—from the first-line software managers through the mid-level software managers to the project software manager—must plan (i.e., budget), track, and control costs within their own area(s) of responsibility. The subcontractor's management also prepares plans for the work assigned to its organization and submits them to the project software manager.

Once planning is completed, each manager can start drawing Figure 4.4-1. The software manager begins by drawing in the budgeted cost at completion. This represents the total cost of the software effort in the current plan. Next the cumulative BCWS is added. This represents the baseline against which project performance is compared each reporting period. This line extends from the beginning to the end of the software project. When the software project ends, the BCWS equals the budgeted cost at completion.

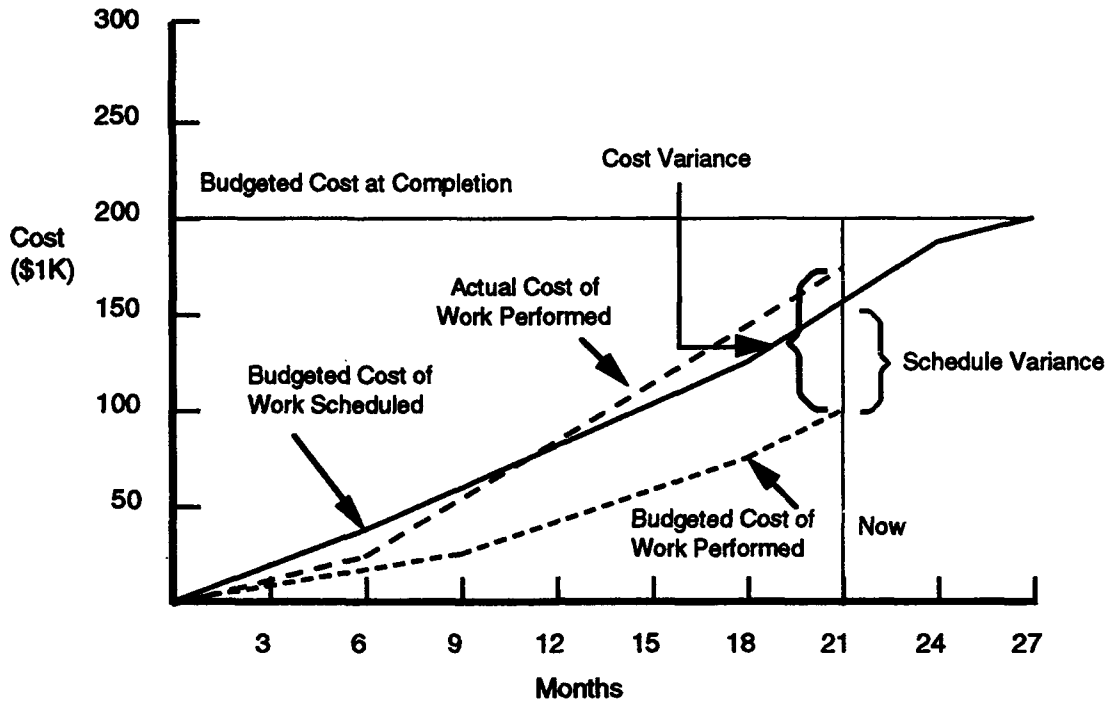


Figure 4.4-1. Cost/Schedule Status Report

For a reporting period, each software manager determines what work was performed and the cost of performing that work. (This information may be gathered through the cost accounting system.) The work actually performed may not be the work that has been scheduled, nor may the personnel who were originally scheduled to perform the work be the personnel who actually performed the work; thus, the need for BCWP and ACWP. During each reporting period, the cumulative BCWP and ACWP are added to the figure.

Each software manager and the project software manager can now compare the actuals (in this case, ACWP and BCWP) with the plan (BCWS). By comparing the BCWP with the BCWS, the manager determines how the project is performing with respect to the work scheduled. The schedule variance indicates whether the costs associated with the work are incurred at the rate scheduled. Note that the schedule variance describes the costs associated with work performed and work scheduled and is associated with the schedule only through the completion of the activities. This indicator is to be used with the progress indicator to determine true schedule status. In Figure 4.4-1, the project is under performing, that is, less work is being performed than scheduled.

By comparing the ACWP with the BCWP, software managers determine how their part of the project is performing with respect to cost. The cost variance, the difference between the BCWP and the ACWP, indicates how fast the project is spending its budget. In Figure 4.4-1, the project is spending more money than budgeted (CV is negative).

The project software manager of the project in Figure 4.4-1 is spending more money than planned, but is achieving less work than planned. The manager needs to determine why the work is not completed on time and why it is costing more to perform the work. Perhaps a more difficult work package is actually being worked on, or more experienced (and higher priced) personnel are being used. To determine additional information on the causes of the problems this project is experiencing, the project software manager needs to look at other indicators, for example, the effort and progress indicators. The project software manager can use a consolidated report to present project status.

Figure 4.4-1 can also be used to predict when the project will be completed based on the trend in the BCWP, and to obtain an estimate at completion based on the trend in the ACWP as shown in Figure 4.4-2.

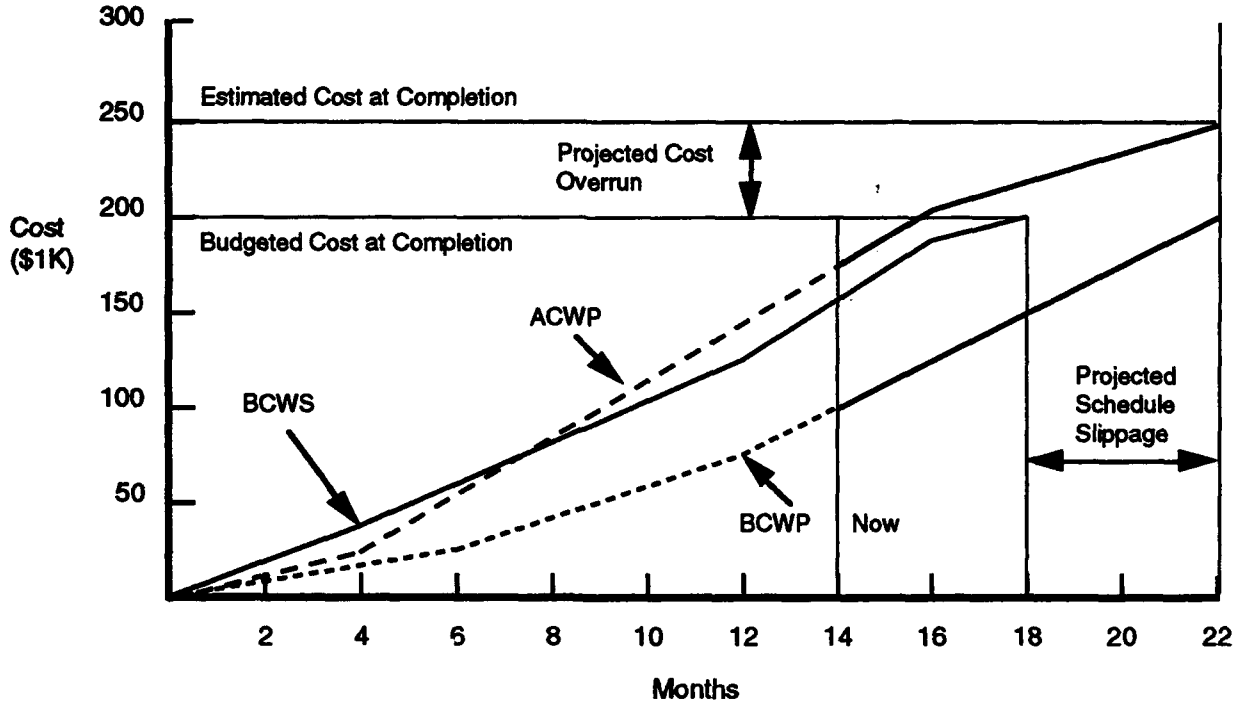


Figure 4.4-2. Cost/Schedule Status Report Showing Project Cost Overrun and Schedule Slippage

In Figure 4.4-2, the ACWP can be extrapolated (and is parallel to the BCWS since the work performed is the same as the work scheduled) to intersect the line obtained by dropping the perpendicular line to the point where the extrapolated BCWP intersects the budgeted cost at completion line to derive an estimated cost at completion. In the figure, if the project was to spend the money at the rate projected by the BCWS for the

remainder of the project, a $50K cost overrun is predicted. Furthermore, a straight line projection of the BCWP indicates that the task will be six months late.

Figure 4.4-3 is a second example of a cost/schedule status report. In this figure, the cost performance to date is favorable, since the ACWP tracks below the BCWP (the cost variance is positive). However, the rate of expenditures increased, as evidenced in the increased slope of ACWP during the past two months (months 5 and 6). The BCWP line stayed relatively flat. These two lines indicate that the project is spending money but not accomplishing the work that has been scheduled. Accordingly, the favorable cost variance that has been accrued in previous months is eroding. There is also an unfavorable schedule situation. The BCWP continues to track below the BCWS line (the schedule variance is negative), indicating that this project is behind schedule. However, since BCWP and BCWS are converging slowly, the amount of slippage is decreasing.

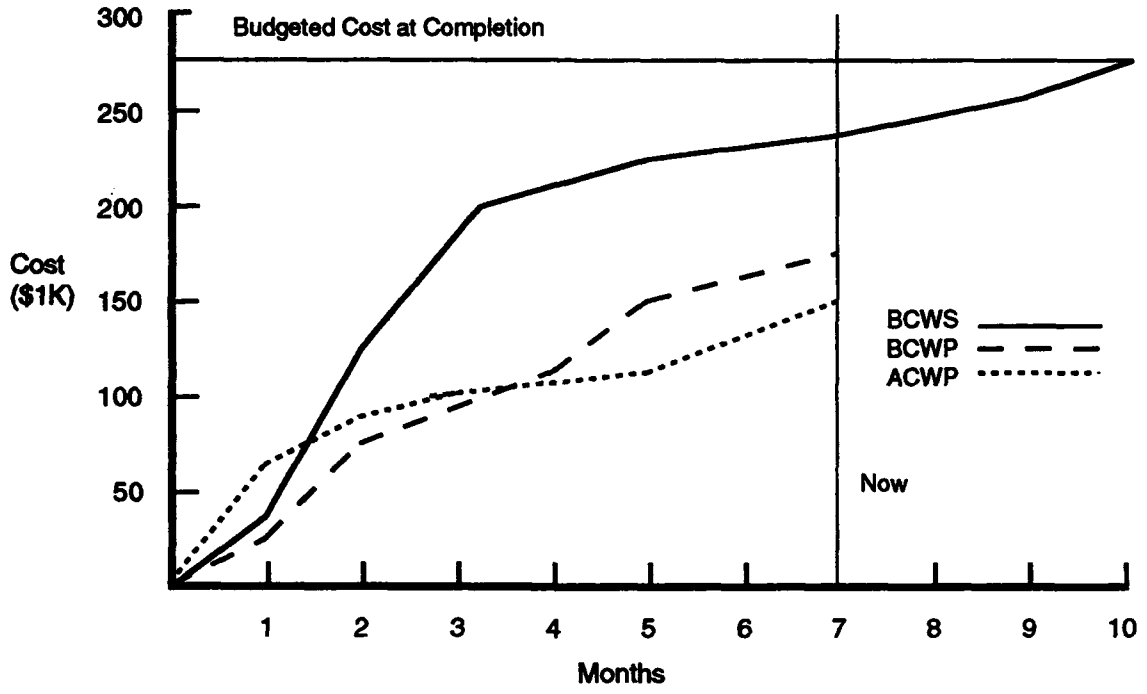


Figure 4.4-3. Cost/Schedule Status Report

In general, the manager looks for the following when analyzing the cost/schedule reports:

- Large or frequent cost variances
- Increasing cost/schedule variances
- Large schedule variances
- Failure to evaluate estimated cost at completion

- Consistently optimistic projections
- Frequent and extensive changes to the plans
- Plans that do not reflect actual conditions and are not being revised

Sources

[AFSC 86] has a discussion of these indicators.

[DoD 80] has a thorough discussion on the basics of the cost/schedule control reporting and provided the definitions for ACWP, BCWP, and BCWS.

[DSDM 89] served as the major source of information in this section.

4.5. Quality

At the Repeatable Level quality indicators are divided among the results of software quality assurance audits, the results of life-cycle reviews with the customer, and the trouble reports written after the implementation team has released the software for testing. Even though there are numerous similarities among these three indicator categories, and even though these indicators are discussed together by Florac et al [Florac 92], they are discussed separately in this document to emphasize that the information is gathered from three distinct activities and that each is used to obtain a indication of the quality of the product and process.

4.5.1. Software Quality Assurance Audit Results

One of the goals of software quality assurance is to improve software quality by monitoring both the software and the development process that produces it. An effective way to accomplish this goal is through a quality audit. A quality audit checks a product or process for compliance with project standards, procedures, and progress reporting policy. When performed by an independent software quality assurance organization, the project software manager is receiving an objective evaluation of the product or process.

Objective of the Software Quality Assurance Audit Results Indicators

To provide project software management with an independent evaluation of the quality of the product and/or adherence of the project staff to project requirements, standards, and procedures.

Indicators

- Trends in the number, type, and severity of noncompliance issues found during an audit
- Trends in the rate at which the noncompliance issues are being addressed
- Trends in the rate at which the noncompliance issues are being closed

Key Process Area Goals Addressed

Software Quality Assurance:

- Compliance of the software product and software process with applicable standards, procedures, and product requirements is independently confirmed.
- When there are compliance problems, management is aware of them.

Life-Cycle Stages: All

Users

Mid-level and higher-level software management

Users' Questions

- Are audits conducted by an independent software quality assurance group for each step of the software development process?
- Are standards and procedures applied on the software project?
- Are project personnel applying the standards and procedures correctly?
- Are project personnel following the standard processes (where defined)?
- Are independent audits conducted for the software subcontractor to ensure compliance with the software development plan?
- Are the noncompliance issues being addressed in an appropriate manner?
- Are the noncompliance issues being addressed in a timely manner?

Input

- Number of noncompliance issues:
  - Total
  - Open
  - Closed
- For each noncompliance issue:
  - Date opened
  - Type (e.g., a noncompliance issue of the product to a standard or a noncompliance issue to a process)
  - Severity (degree of noncompliance issue, e.g., product cannot be delivered as is; product must be corrected by next release; recommend process change)
  - Date closed

Interpretation

The project software manager is interested in the number of noncompliance issues open at any one time and the speed with which the noncompliance issues are addressed.

Figure 4.5.1-1 is a plot of the number of noncompliance issues open against time. The figure shows that noncompliance issues are identified in relation to a discrete event, the audit, and it also indicates progress in addressing the noncompliance issues.

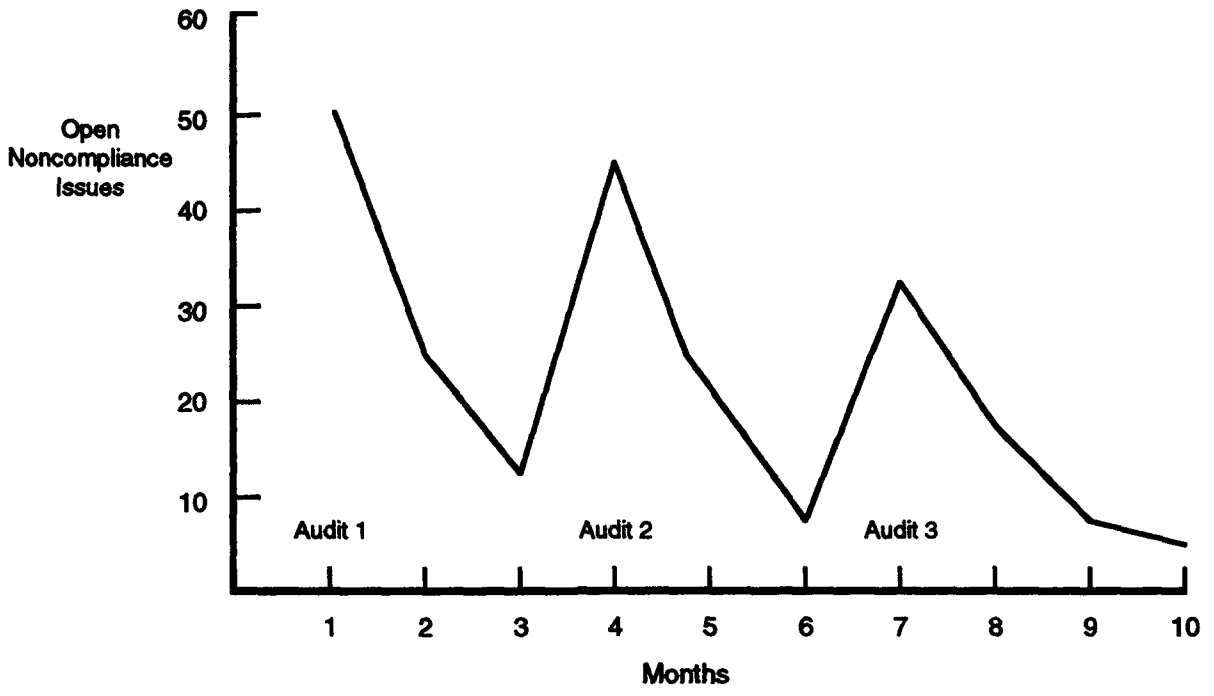


Figure 4.5.1-1. Total Number of Noncompliance Issues Open Over Time

In Figure 4.5.1-1, three audits occurred: one at month 1, one at month 4, and one at month 7. The noncompliance issues are being addressed fairly rapidly by project personnel after each audit, but not all noncompliance issues are closed before the next audit. The number of noncompliance issues reported at each audit also decreases. The audits upon which the figure is based do not have to be of the same product or process. If data were obtained from different products produced by the same personnel using the same standard, and if they were normalized to account for different product sizes, then the manager could conclude that the personnel are becoming more proficient in the application of the relevant standards. Ideally, the manager can expect fewer noncompliance issues with time since the personnel are more familiar with the standards and procedures.

One reason that the noncompliance issues in Figure 4.5.1-1 may be addressed fairly rapidly is that they can involve minor issues, for example, a minor deviation from a coding or documentation standard. This can be clarified by plotting the severity of the noncompliance issue. If the number of noncompliance issues is small in each of the more severe categories, they can be combined to make a more meaningful graph.

The manager is unable to determine from Figure 4.5.1-1 how many of the open noncompliance issues in month 6 are from the first audit and the second. That information is contained in Figure 4.5.1-2, which shows the status of noncompliance issues related to each individual audit. This figure shows the total number of

noncompliance issues for each audit as well as the information displayed in Figure 4.5.1-1: the number of noncompliance issues open at any time and the rate at which they are closed.

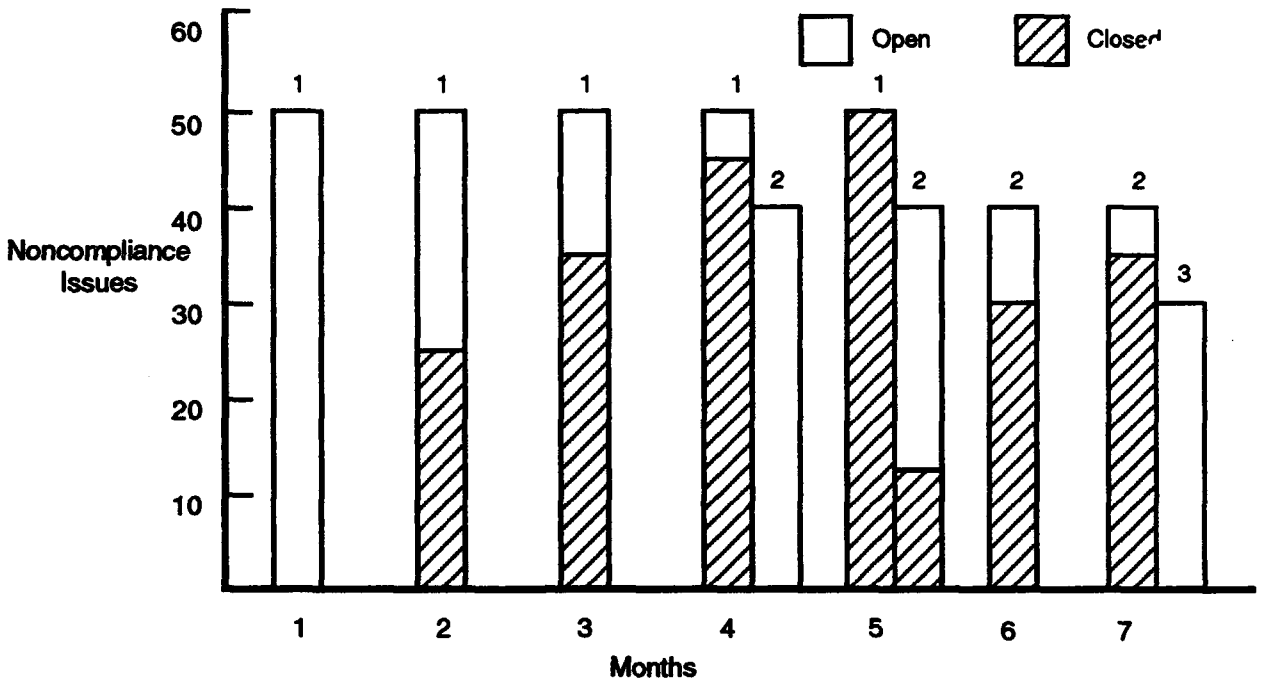


Figure 4.5.1-2. Number of Noncompliance Issues by Individual Audit

An alternative to Figure 4.5.1-2 is to plot the data from each audit on Figure 4.5.1-1 on separate graphs.

Software quality assurance personnel prepare reports summarizing the results of the audit. The report is distributed to the project software manager and other appropriate managers. Managers receive a report only if it applies to their area of responsibility. When severe quality problems exist and/or the noncompliance issues are not being addressed adequately, the software quality manager reports the noncompliance issues and their status to senior management.

Sources

[Pfleeger 89] and [Florac 92] discuss the tracking of problems found in a variety of life-cycle stages. The software quality assurance audit results indicators are an extension of that discussion.

4.5.2. Review Results

Of the many reviews held on a project, those held with the client are among the most important. These formal reviews ensure coordination of the technical and business aspects of the project, make all participating groups aware of the project status, and either resolve or escalate key issues [Humphrey 89]. These reviews are major events in the life of the project and are not peer reviews or the periodic status reviews held with the customer to review cost and schedule issues. In the Department of Defense 2167A model, these reviews are the end-of-phase reviews wherein the customer gives the approval to proceed to the next life-cycle phase. Also, these reviews can be used to establish a baseline from which all further work proceeds.

At these reviews, action items are levied which are used in this indicator category. In this document an action item is defined as any review discrepancy, clarification, or issue that must be resolved by the project or the customer. Issues raised during peer reviews are NOT included in the review results indicator. Peer review moderators must track the action items originating in the peer reviews, but the shortened time scales within the peer review process render the formalism discussed in this section impractical.

Objective of the Review Results Indicators

To provide software project management, senior management, and the customer with *the status of action items originating during a life-cycle review.*

Indicators

- Trends in the number, type, and priority of action items recorded during a review
- Trends in the rate at which the action items are being addressed

Key Process Area Goals Addressed

Software Project Tracking and Oversight:

- Actual results and performance of the software project are tracked against documented and approved plans.
- Corrective actions are taken when the actual results and performance of the software project deviate significantly from the plans.
- Changes to software commitments are understood and agreed to by all affected groups and individuals.

Software Quality Assurance:

- Compliance of the software product and software process with applicable standards, procedures, and product requirements is independently confirmed.
- When there are compliance problems, management is aware of them.

Software Configuration Management:

- Controlled and stable baselines are established for planning, managing, and building the system.

Life-cycle stages: All

Users

- Project software manager
- Software engineering process group
- Customer
- Senior management
- Software quality assurance
- Software engineering

Users' Questions

- Is the review process being followed?
- Are the action items being handled in a timely manner?
- How are the number and types of open action items going to impact the cost, schedule, and resources?

Input

- Number of action items:
  - Total
  - Open
  - Closed
- For each action item:
  - Date opened
  - Type (e.g., documentation change, additional analysis required, resolution of a to-be-determined item, process)
  - Priority (e.g., relative ranking of importance of the action item)
  - Date closed

Interpretation

The project software manager is interested in the number of action items open at any time and the speed with which the action items are addressed.

Figure 4.5.2-1 is a plot of the number of action items open against time. The figure shows that action items are issued in relation to a discrete event, the review, and it also shows progress in addressing the action items.

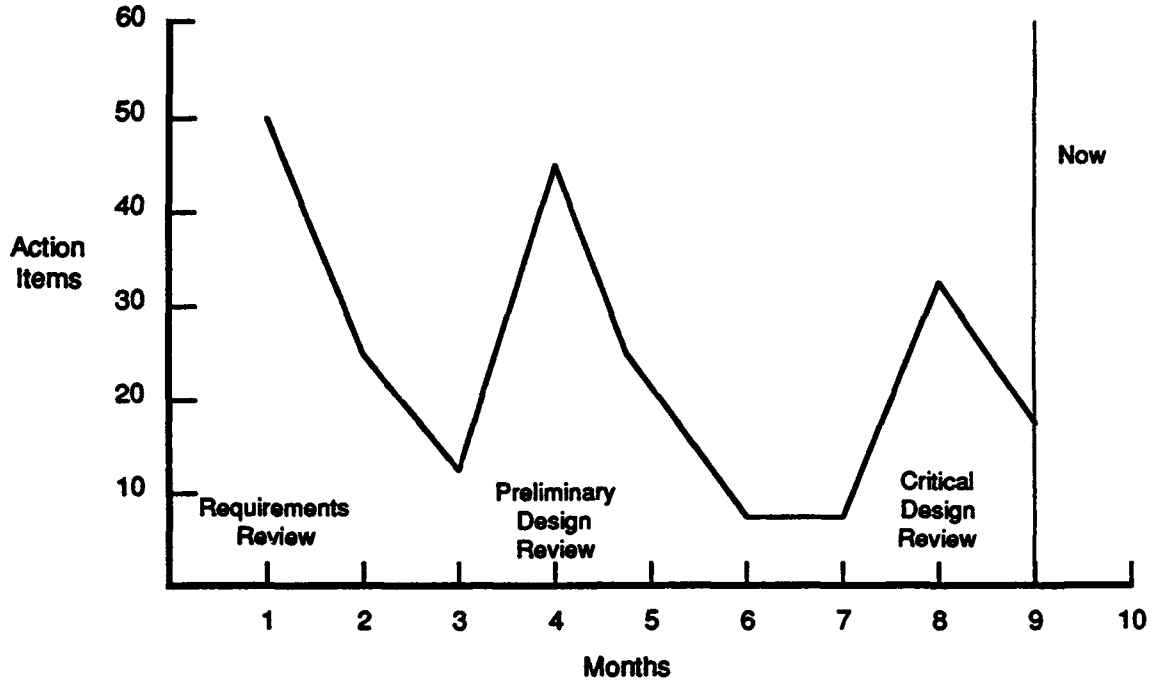


Figure 4.5.2-1. Total Number of Action Items Open Over Time

In Figure 4.5.2-1, three reviews occurred: a requirements review (RR) at month 1, a preliminary design review (PDR) at month 4, and a critical design review (CDR) at month 8. The action items are being addressed fairly rapidly by project personnel after each review, but not all action items are closed before the next review. Action items open for more than sixty days should be closely examined for schedule impact, especially if the action item has been classified as one of higher priority.

The figure also shows that the number of action items reported at each review is decreasing. The manager might conclude that the personnel are better prepared at each review, perhaps as the result of a good understanding of the requirements and the design process. In general, the number of action items reflects on the quality of the products reviewed and the review itself.

One reason that the action items in Figure 4.5.2-1 may be addressed fairly rapidly is that they can involve minor issues. This can be clarified if plots are made by the priority of the action items. If the number of action items is small in each of the higher priority categories, they can be combined to make a more meaningful graph. Project personnel should be addressing the higher priority items before the lower-level items.

In Figure 4.5.2-1, the manager cannot tell how many of the open action items in month 6 are from the RR or the PDR. That information is contained in Figure 4.5.2-2 which shows the status of action items related to each individual review.

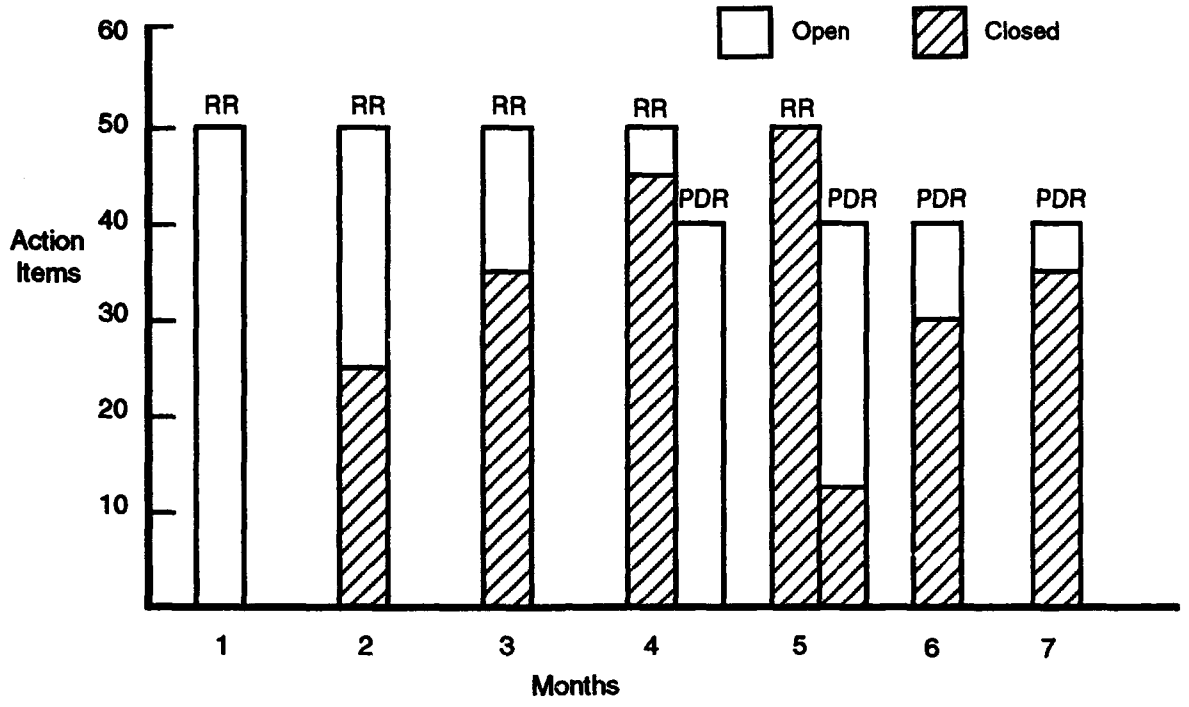


Figure 4.5.2-2. Number of Action Items by Individual Review

Figure 4.5.2-2 shows the total number of action items for each review as well as the information displayed in Figure 4.5.2-1: the number of action items open at any time and the rate at which they are closed.

An alternative to Figure 4.5.2-2 is to plot the data from each review shown in Figure 4.5.2-1 on separate graphs. The manager may also want to separate process action items from those involving product as they are addressed by different groups. The action items can be separated by their type.

Software quality assurance personnel prepare reports summarizing the results of the review. These reports are distributed to the project software manager, the customer, and other appropriate managers.

Sources

[AFSC 86] discusses tracking action items in its requirements definition and stability indicators.

[Florac 92] discusses tracking action items in a general discussion of a problem management system.

[Rozum 92] discusses tracking action items according to a software defects metric.

[Schultz 88] discusses tracking action items in his software volatility metric.

4.5.3. Trouble Reports

Trouble reports provide an indication of the quality of the product not only by their number, but also by the rate at which they are written. The number of trouble reports also reflects the amount of rework. A trouble report is a document (electronic or hard copy) used to recognize, record, track, and close anomalies detected in the software and its accompanying documentation. Trouble reports are often referred to as problem reports, discrepancy reports, anomaly reports, etc. In this document, trouble reports are restricted to those written during the integration and test, and the acceptance test activities, that is, those activities conducted after the implementation team turns the software and its accompanying documentation over to an independent testing team.

Terminology is important in the discussion of problems and defects. In this document, the following definitions are used:

- A defect is a product's inconsistency with its specification. Examples include such things as omissions and imperfections found in software during the early life-cycle phases and faults in software that is sufficiently mature for test or operation.
- An error is a human action that produces an incorrect result [IEEE 610].
- A fault is an incorrect step, process, or data definition in a computer program [IEEE 610]. It is synonymous with bug.
- A failure is the inability of a system or component to perform its required functions within specified performance requirements [IEEE 610].
- A problem is an unsettled question arising from a situation where it appears that a change to the software, its documentation, or related hardware is necessary for successful test or operation of the system [IEEE P1044].

Objective of the Trouble Reports Indicators

To provide software managers with insight into the quality of the product, the software reliability, and the effectiveness of testing.

Indicators

- Trends in the following:
  - Number, type, and severity of the trouble reports
  - Trouble report density, that is, the number of trouble reports per unit size
  - Rate at which trouble reports are being addressed
- Relationship between the number of trouble reports and the number of test cases passed

Key Process Area Goals Addressed

Software Project Tracking and Oversight:

- Corrective actions are taken when the actual results and performance of the software project deviate significantly from the plans.

Software Configuration Management:

- Controlled and stable baselines are established for planning, managing, and building the system.
- The integrity of the system's configuration is controlled over time.
- The status and content of the software baselines are known.

Life-Cycle Stages

Integration and test, and acceptance testing

Users

- Project software manager
- Software quality assurance personnel
- Software testing manager
- First-line and mid-level software development managers

Users' Questions

- Are any particular test phases exhibiting unusual characteristics?
- Will undetected or unresolved problems in the product lead to more problems in the next life-cycle stage?
- Does the number of trouble reports indicate that the software product should be reworked before proceeding to the next life-cycle stage?
- Is the testing activity complete?
- Does the quality of the product indicate that the product is ready for release to the customer?
- Are project personnel addressing the trouble reports in a timely manner?

Input

- Number of trouble reports:
  - Total
  - Open
  - Closed
- For each trouble report:
  - Date opened
  - Date closed
  - Date trouble report evaluated

  - Type
  - Severity
- Number of test cases:
  - Scheduled
  - Passed
- Product size

Interpretation

Numerous graphs can be generated. All are basically a plot of the number of trouble reports against time. Figures that are recommended at the Repeatable Level are:

- Total number of trouble reports against time
- Number of trouble reports open against time
- Number of trouble reports closed against time
- Number of unevaluated trouble reports against time

The above items can be plotted by type or severity or combinations of types or severities. The manager can also use plots of the number of trouble reports per unit size against time and the number of trouble reports against the number of test cases passed. If there are multiple computer software configuration items (CSCI), the data can be plotted for each CSCI and/or release of the CSCI.

Figure 4.5.3-1 shows the number of total, open, and closed trouble reports against time. The total number of trouble reports is the sum of the open and closed trouble reports. It provides an indication of the quality of the product.

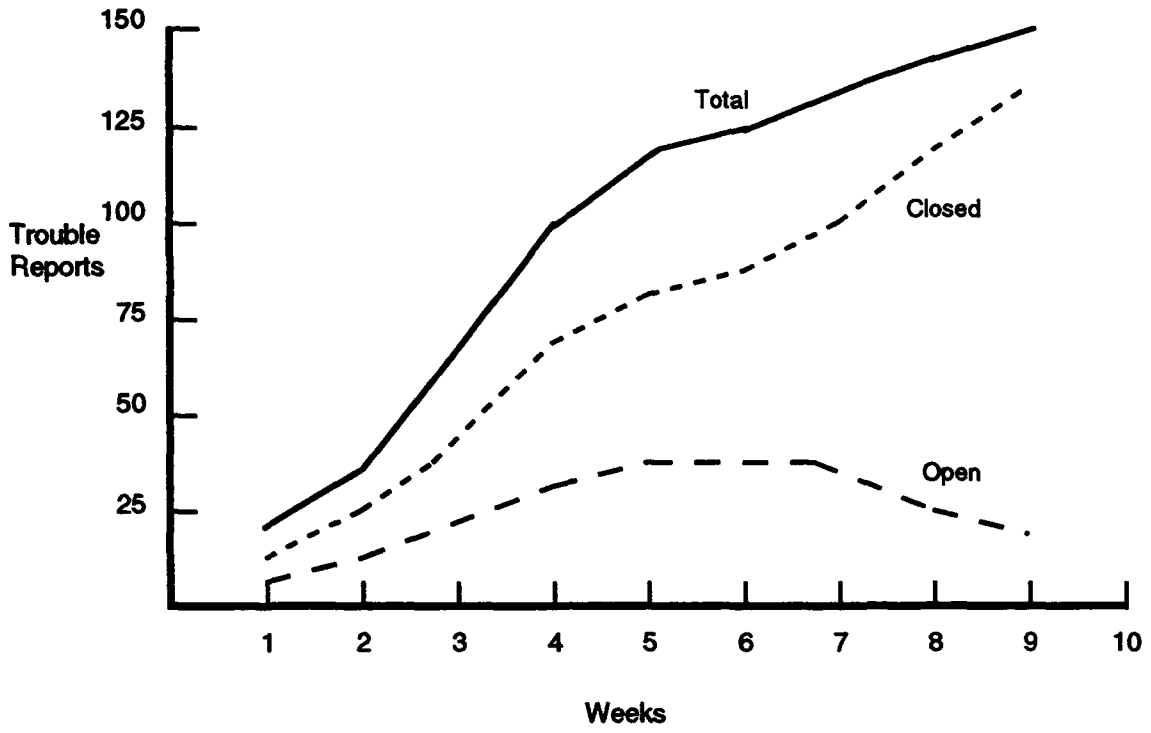


Figure 4.5.3-1. Number of Total, Open, and Closed Trouble Reports

In the analysis of Figure 4.5.3-1, the manager looks for the following:

- The total number of trouble reports
- The number of open trouble reports
- The rate at which trouble reports are written

During project planning, the manager estimates the number of trouble reports expected, based on historical data, and ensures that this estimate is consistent with cost and schedule estimates. If the actual number exceeds the estimate, other estimates, such as cost and schedule, may become invalid. The number of trouble reports reflects the amount of rework that must be completed before the product can be delivered to the customer. If the amount of rework was not accurately estimated, cost and schedule impacts will occur.

Ideally, the number of trouble reports open at any time should remain relatively close to zero. If it deviates substantially from zero or continues to grow, the manager needs to reevaluate the allocation of staff resources. Depending on the severity of the open trouble reports, the manager may redirect staff to work on trouble report closure instead of continued implementation. Figure 4.5.3-4 can be used in conjunction with Figure 4.5.3-1 to aid in that decision. An alternate way of analyzing the same data is the rate of convergence between the closed and total numbers of trouble reports. The more rapid

the convergence of the total and closed trouble reports, the more quickly the project is addressing trouble reports.

The rate of newly created trouble reports should decrease as testing progresses, that is, the curve of total trouble reports decreases over time. Ideally, as testing progresses, it takes the test team longer and longer to discover new problems since the problems remaining are more subtle or represent processing paths not frequently run. Early testing discovers the more common defects. If the number of trouble reports written does not decrease each reporting period, there may be a serious problem with the quality of the product or testing strategy, and the manager needs to evaluate whether the product should be delivered to the customer.

Figure 4.5.3-2 shows the number of trouble reports opened and closed during each reporting period.

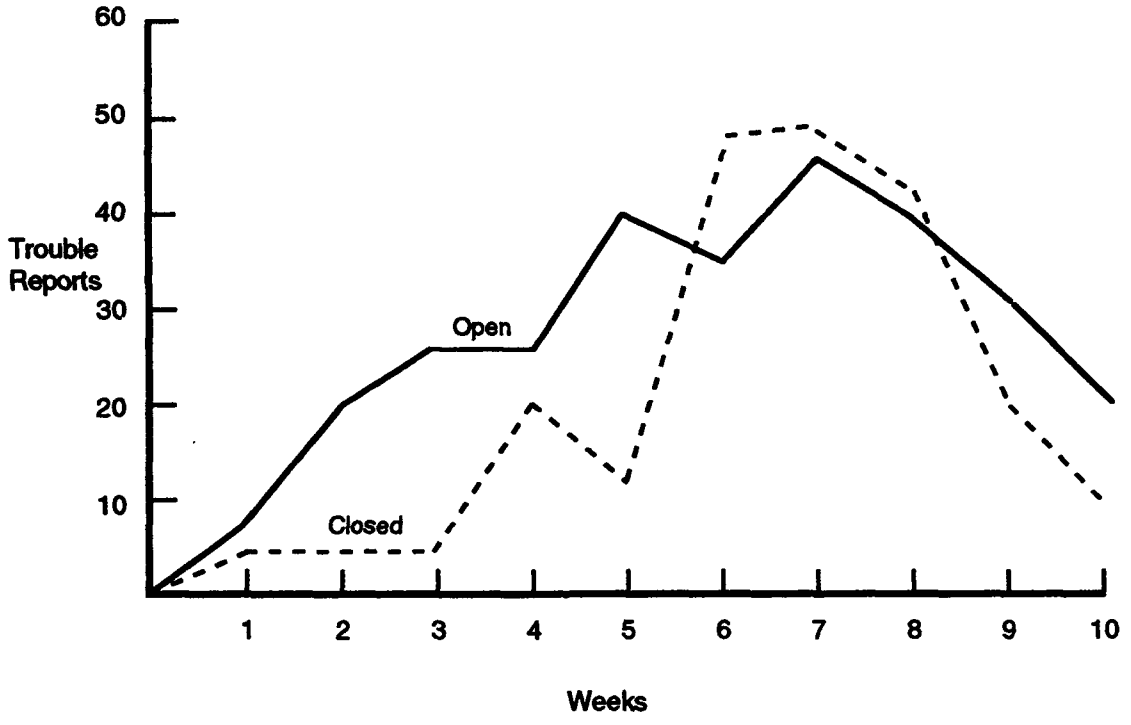


Figure 4.5.3-2. Number of Opened and Closed Trouble Reports in Reporting Period

The number of open trouble reports in the figure indicates:

- Product quality
- The amount of rework that is necessary due to newly created trouble reports

- Test team and external client satisfaction with product (the fewer the number of defects, presumably the more satisfied they are with the product)
- The cost and time required for trouble report evaluations

The number of closed trouble reports indicates:

- The application of resources to correct the problems
- The need for further investigation. If the number of closed trouble reports is low for several consecutive reporting period, the manager needs to determine why they are not being closed, especially if new trouble reports are being written.

The number of closed trouble reports will tend to fluctuate more than the number open since problems are generally corrected in large manageable groups. In addition, if the plot extends far enough in time, the number of open trouble reports will rise sharply whenever a new activity is started, for instance, at the start of a new test for each release, since the obvious problems will surface quickly.

Figure 4.5.3-3 shows the percentage of trouble reports that have a high severity. Ideally, the project wants this percentage to be as close to zero as possible, and the percentage should also steadily decline. The manager should determine the cause of any spike that occurs. If the higher severity percentage declines sharply while the total number of trouble reports remains high, this may indicate that the test team is now detecting more user-oriented defects.

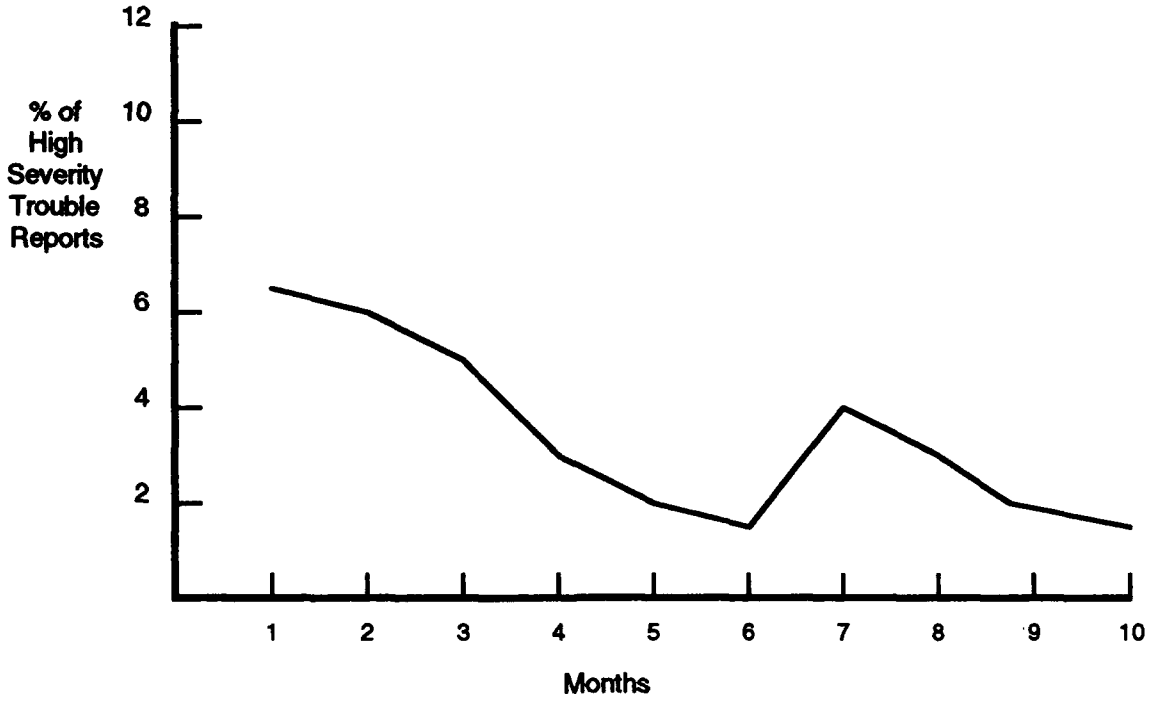


Figure 4.5.3-3. Percentage of High Severity Trouble Reports over Time

Figure 4.5.3-4 shows the number of unevaluated trouble reports for each reporting period. This number varies with the number of trouble reports generated and the resources applied to evaluate them. The more unevaluated trouble reports, the less knowledge the manager has of the amount of resources required to fix inherent problems and the effect of the problems on the schedule. The figure shows the manager when extra resources are needed to reduce the number of unevaluated reports and to minimize the difference between the total number of trouble reports and the number of evaluated ones.

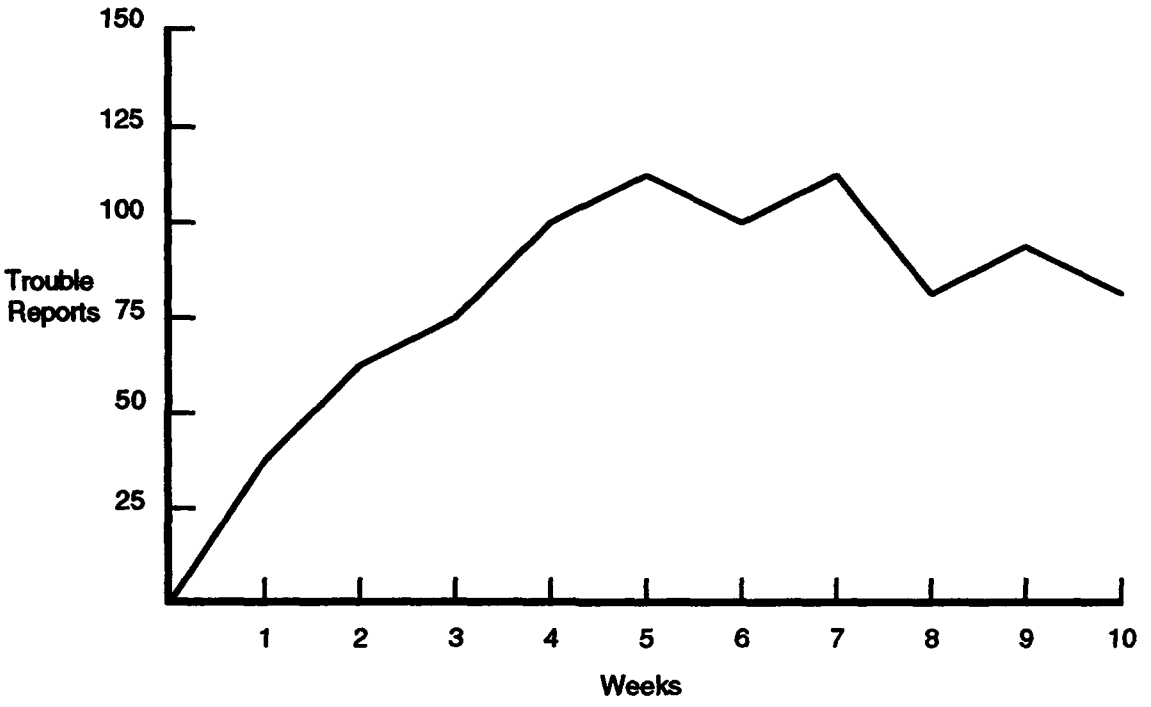


Figure 4.5.3-4. Number of Unevaluated Trouble Reports

Figure 4.5.3-5 shows the number of trouble reports per size over time, that is, the figure is a plot of the number of trouble reports originated to date divided by the size of the software under test. This figure is plotted for every release and for the software system as a whole or for each CSCI. The size of the product is nearly constant since any changes in the size of the product result from fixes to problems and should be small compared to the product size. The number of trouble reports is cumulative and grows each time a trouble report is created. Hence, this ratio will always increase. As testing progresses, the curve should flatten out as fewer trouble reports are created.

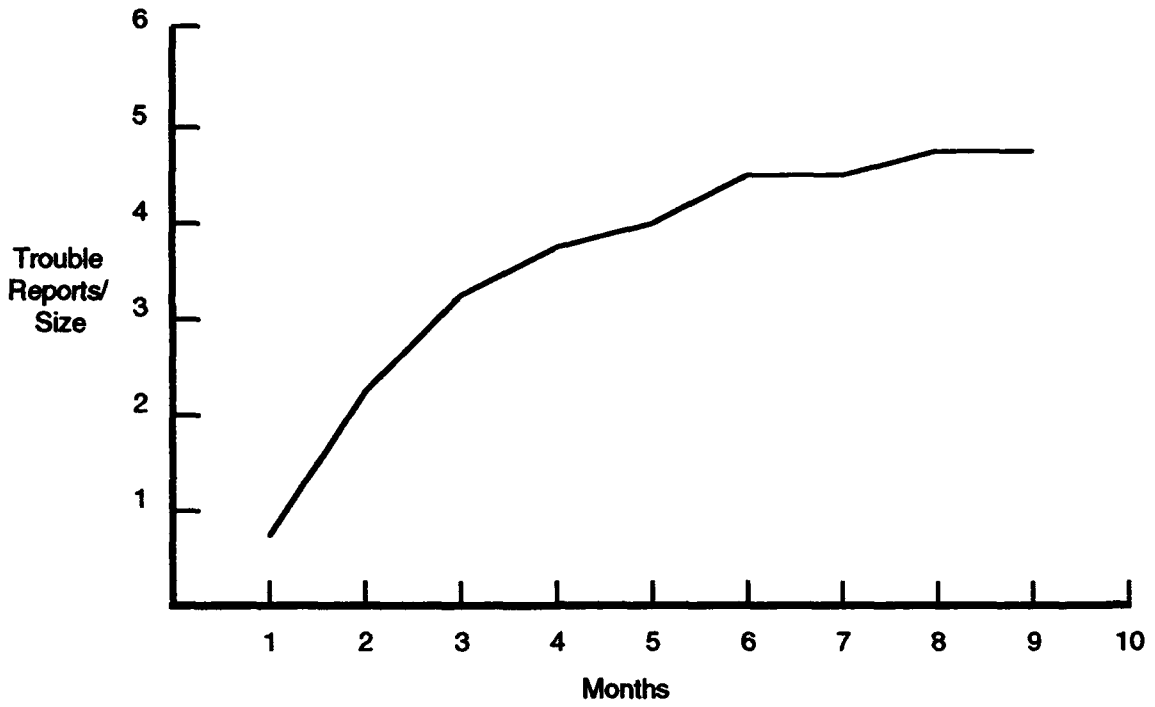


Figure 4.5.3-5. Trouble Report Density

Ideally, the defect density should be plotted rather than the trouble report density, since several different trouble reports can lead to the same defect. At the Repeatable Level, however, the defect data may not be suitable for such an analysis. Accordingly, the trouble report density is used. At the Defined and higher maturity levels, it is appropriate to use the defect density.

Figure 4.5.3-5 is plotted for a single release. If desired, the figure can be continued over time. In this case, the observed trouble report density will drop each time a new build or release is delivered to the test group since the size of product under testing increased. After the initial decrease, the density will resume increasing as new trouble reports are generated. The observed density will approach, or oscillate around, the value characteristic of the quality of the developed product.

When Figure 4.5.3-5 is used in conjunction with a plot of the number of test cases created against the number of test cases passed (see Figure 4.5.3-6), the manager can predict the final quality of the product and can determine whether and when the product is ready for delivery to the customer.

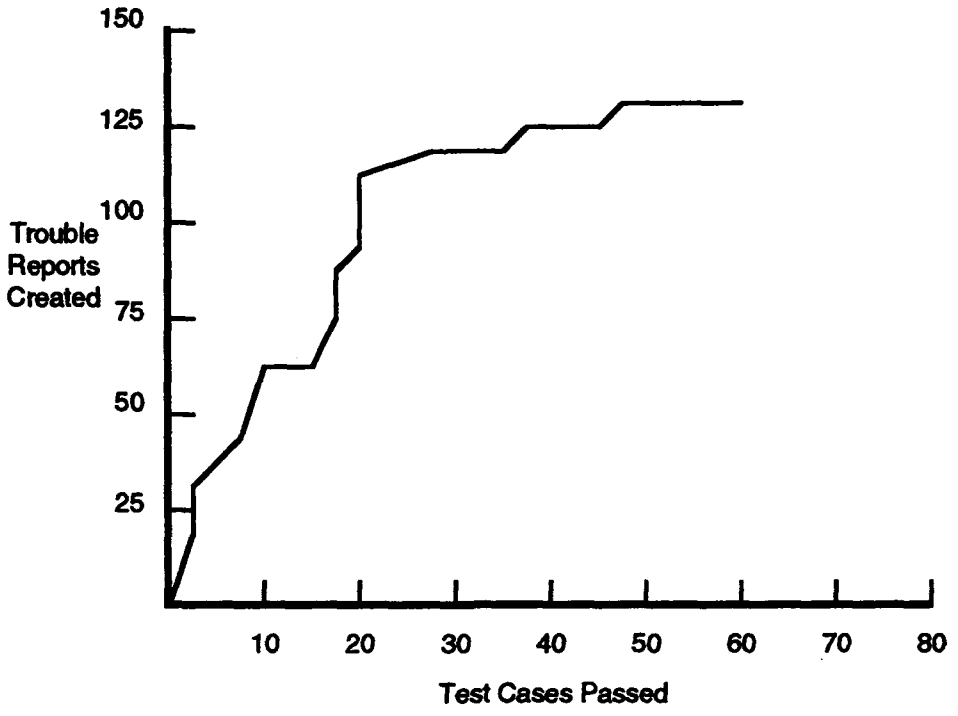


Figure 4.5.3-6. Trouble Reports Created Against Test Cases

Figure 4.5.3-6 shows the number of trouble reports created versus testing progress. The manager uses this figure to determine the adequacy of testing. As testing progresses, the yield in terms of new trouble reports created with each new test case declines. The curve should flatten out since it requires more test cases to detect new problems. If the curve continues to rise steadily, more test cases may be required. As long as the curve continues to rise the product is not ready for delivery to the customer.

This figure is more appropriately drawn for each severity class or combination of severities. It is often difficult to deliver a product with no trouble reports open. It is important that all the higher severity trouble reports have been addressed prior to delivery. If Figure 4.5.3-5 is drawn for the total number of trouble reports, the lower severity problems may mask persistent higher severity problems.

The project determines the frequency of the reports and the length of the reporting period. Due to the shortness of the testing period and the dynamics involved between the test team and the implementation team that analyzes the trouble report and fixes the defect, the information is much more useful if provided on a weekly basis, rather than a biweekly or monthly basis.

Sources

[AFSC 87], [Buckley 90], [Card 90], [Decker 91], [Grady 87], [IEEE 1061], [Landis 90], [Pfleeger 89], and [Rozum 92] discuss trouble reports.

[Florac 92] has a thorough discussion of trouble reports and serves as the main source of information for this section.

4.6. Stability

At the Repeatable Level the stability indicators concentrate on the stability of the requirements and the size. Requirements stability is concerned with the number of changes to the requirements and the amount of information that still needs to be determined with regard to the requirements. Size stability is concerned with the stability of code size and size estimates.

4.6.1. Requirements Stability

The lack of requirements stability can lead to poor product quality, increased project cost, and/or lengthened project schedule. The requirements stability indicators consist of trend charts that show the total number of requirements, the cumulative number of changes, and the number of to-be-determineds (TBDs) over time.[1] Landis et al report that a large number of TBDs in the requirements and specifications, combined with a large number of requirements changes, have caused systems in their environment to grow up to forty percent larger than size estimates made at preliminary design review [Landis 90].

Objective of the Requirements Stability Indicators

To provide the software manager with visibility into whether requirements changes are responsible for cost overruns, schedule delays, and decreased product quality.

Indicators

- Trends in the total number of requirements changes
- Trends in the number of TBDs

Key Process Area Goals Addressed

Requirements Management:

- The system requirements allocated to software provide a clearly stated, verifiable, and testable foundation for software engineering and software management.
- The allocated requirements define the scope of the software effort.
- The allocated requirements and changes to the allocated requirements are incorporated into the software plans, products, and activities in an orderly manner.

---

[1] The organization must have a definition for what constitutes a unit of requirements such as a numbered paragraph containing the word "shall."

Software Project Tracking and Oversight:

- Changes to software commitments are understood and agreed to by all affected groups and individuals.

Software Configuration Management:

- Controlled and stable baselines are established for planning, managing, and building the system.
- The integrity of the system's configuration is controlled over time.
- The status and content of the software baselines are known.

Life-Cycle Stages: All, but most important during requirements and design.

Users

- Software engineering manager for controlling and monitoring of requirements.
- Software engineering process group for sources of and reasons for instability.

Users' Questions

- Is the number of changes to the requirements manageable?
- Are the requirements scheduled for implementation in a particular release actually addressed as planned?
- Is the number of changes to requirements decreasing with time?
- Is the number of TBDs preventing satisfactory completion of the product?
- Is the number of TBDs decreasing with time, that is, are the TBDs being resolved in a timely manner?

Input

- Total number of requirements
- Number of requirements changes:
  - Proposed
  - Open
  - Approved
  - Incorporated into baseline
- For each requirements change:
  - The computer software configuration item(s) (CSCI) affected
  - Major source of request (customer, software engineering, etc.)
  - Requirement type (functional, performance, etc.)
- Number of TBDs in requirements specifications
- Number of requirements scheduled for each software build or release

Interpretation

Figure 4.6.1-1 shows the current total number of requirements, the cumulative number of changes to requirements, and the number of TBDs remaining in the requirements over time for the entire project. Similar plots can be made for each CSCI. Such a graph shows the distribution of changes over the CSCIs and highlights the CSCI(s) responsible for the majority of the changes, if the changes are not equally distributed across the CSCIs.

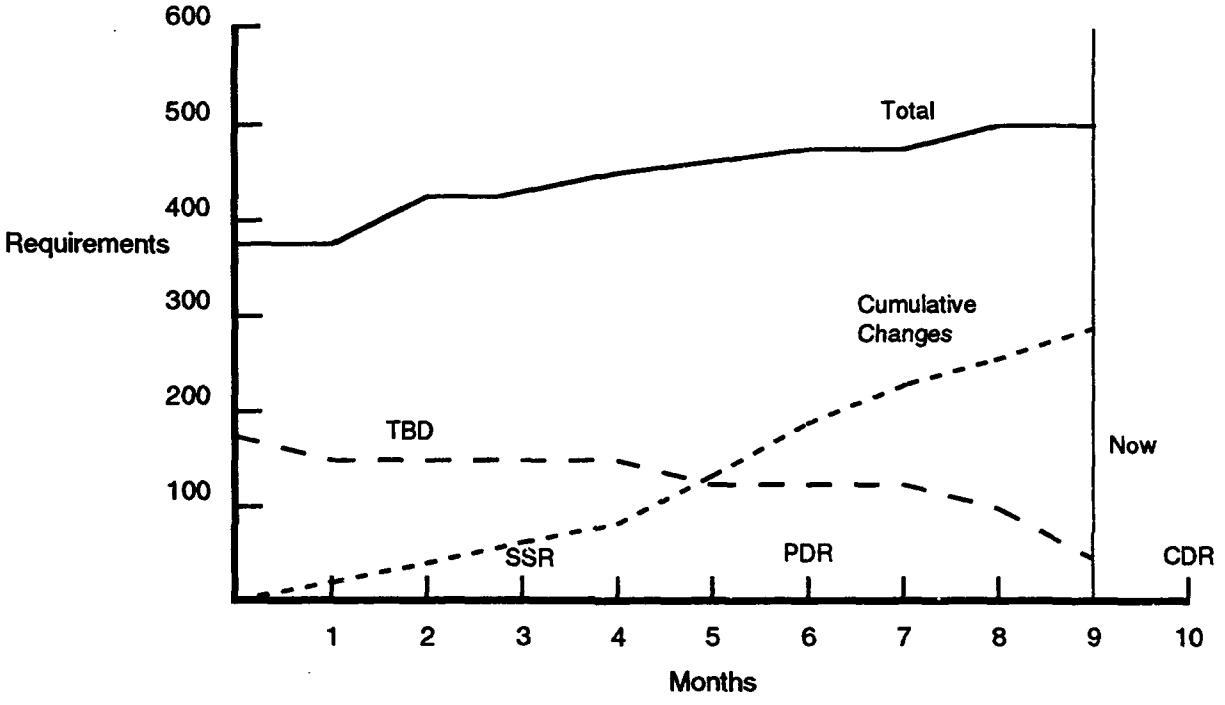


Figure 4.6.1-1. Current Total Number of Requirements, Cumulative Changes, and Number of TBDs Against Time

Typically, the plot of cumulative changes can be expected to rise more steeply from the software specification review (SSR) to preliminary design review (PDR) and to level off after the critical design review (CDR). This is a consequence of the activities occurring on the project. During requirements analysis, the software engineering group is analyzing the system requirements allocated to software. This analysis can lead to changes in the software requirements. By the time the project reaches the CDR milestone, requirements issues should be fully addressed. Changes can still occur, but the frequency of the changes and the number of change requests should fall off dramatically. If this does not occur, it may be necessary to delay the implementation of the CSCI or the portion of the CSCI that is responsible for the high number of requirements changes until the requirements are stabilized. If significant changes occur in requirements after CDR, the manager runs the risk of substantial rework in previously

developed products. Also, any changes occurring after CDR can be expected to have a significant schedule impact, even if the change is the deletion of a requirement.

Similarly, the number of TBDs should decrease at each review and ideally be zero by CDR. The number should decrease since the analysis of requirements leads to the resolution of the items that need to be determined. If the number does not decrease, the manager must be aware of the fact that the longer it takes to eliminate a TBD, the greater the likelihood of rework on that requirement with a corresponding impact on cost and schedule. The manager's risk increases the longer each TBD remains unresolved.

Figure 4.6.1-2 shows requirements changes broken down by type. The different types of requirements changes impact projects differently. Added requirements tend to increase the size of the product and the effort and cost. Modified and deleted requirements tend to affect the quality of the product, particularly if made late in the development life cycle.

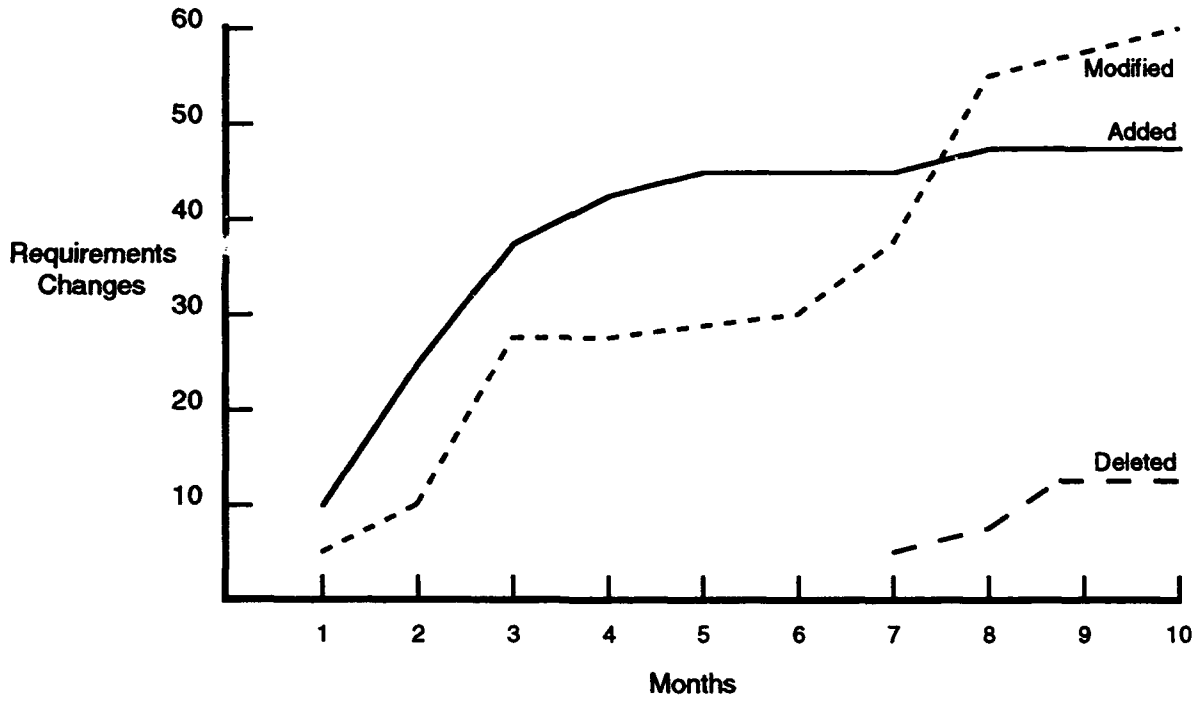


Figure 4.6.1-2. Types of Requirements Changes

Figure 4.6.1-3 shows the number of requirements changes recorded during each reporting period. It displays more emphatically the fact that significant changes can occur to system requirements during the requirements definition and preliminary design activities and that the number of changes decreases during detailed design.

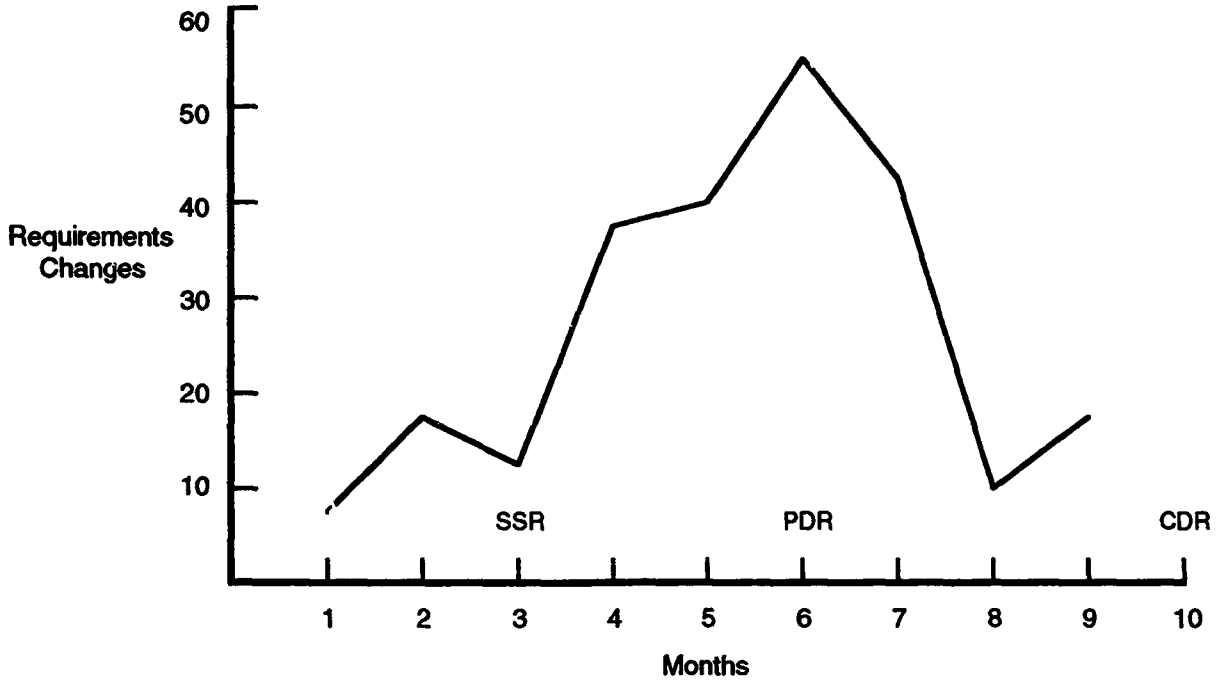


Figure 4.6.1-3. Number of Requirement Changes Recorded During Each Reporting Period

Figure 4.6.1-4 shows the number of requirements planned for implementation in each software release and the number of requirements actually implemented. The actual number of requirements satisfied should stay close to the planned number of requirements satisfied in each build or release. The number in any one build or release may grow as a result of the addition of requirements.

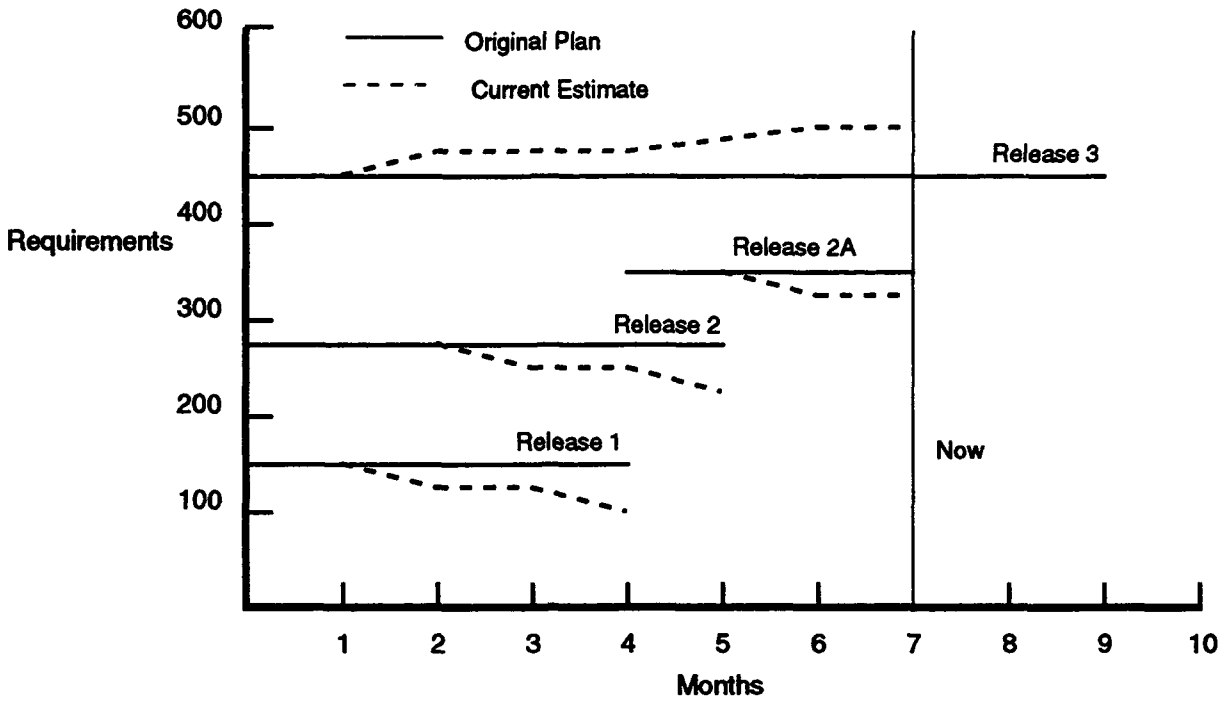


Figure 4.6.1-4. Number of Requirements per Release

The project shown in the figure may be experiencing schedule trouble. The number of requirements satisfied in the first release did not match the plan, and the same trend has occurred for the second release. The project planned on satisfying these requirements in release 2A but, again, the project is not meeting its goals. The number of requirements for release 3 is increasing, but past history indicates a schedule slippage for the project.

Sources

[AFSC 86] has a similar indicator that uses software size.

[Decker 91], [Pfleeger 89], and [Schultz 88] discuss requirements stability.

[Grady 87] uses a requirements stability as an input to Hewlett-Packard's difficulty metric.

[Landis 90] discusses two indicators that are related to this but uses software size in the trend chart.

4.6.2. Size Stability

Size is an important input in the planning process. The project software manager needs a good estimate of the size of the software to derive accurate estimates of the effort, cost, and schedule of the project. Changes in the size ripple through the project with effects on effort, cost, and schedule.

The project software manager selects from the approved organizational methods the technique to be used by the project for determining size. There are many units of measure for software size, for example, source lines of code, delivered source instructions, function points, objects, or tokens. While this document makes no recommendation on how to determine size (examples of counting lines of code are given by Park [Park 92]), it does describe how to interpret trends in the size measure.

Objective of the Size Stability Indicators

To provide the project software manager and the project manager with an indication of the completeness and stability of the requirements and of the capability of the implementation staff to produce the software product within the current budget and schedule.

Indicators

- Trends in the code size
- The variation of actual software size from size estimates
- Variation of actual software size from estimated size by build or release

Key Process Area Goals Addressed

Software Project Planning:

- All affected groups and individuals understand the software estimates and plans and commit to support them.
- The software estimates and plans are documented for use in tracking the software activities and commitments.

Software Project Tracking and Oversight:

- Actual results and performance of the software project are tracked against documented and approved plans.
- Corrective actions are taken when the actual results and performance of the software project deviate significantly from the plans.
- Changes to software commitments are understood and agreed to by all affected groups and individuals.

Software Subcontract Management:

- The software standards, procedures, and product requirements for the subcontractor comply with the prime contractor's commitments.

- The prime contractor tracks the subcontractor's actual results and performance against the commitments.

Software Configuration Management:

- Controlled and stable baselines are established for planning, managing, and building the system.
- The integrity of the system's configuration is controlled over time.
- The status and content of the software baselines are known.

Life-Cycle Stages: All

Users

Software engineering and project software managers for controlling and monitoring of project

Users' Questions

- How much have the size estimates changed over time during development?
- How much do the actual values deviate from their estimated values?
- How much does the trend of the actual values affect the development schedule?
- Is the estimated productivity sufficient to allow the completion of added code on schedule or are more staff required?

Input

- Software size (estimates and actual values):
  - Total size
  - By computer software configuration item (CSCI)
  - New code
  - Off-the-shelf code
  - Reused code
- Amount of software scheduled for completion by build or release

Interpretation

Figure 4.6.2-1 shows how the total software size, as well as the new and reused code, changed over time. The amount of generated, converted, or removed code can also be plotted. The growth of software size should reflect requirements stability, if the size estimates are accurate. Any change in size can then be directly linked to a requirements change. In practice, this linkage is not so obvious, but the growth in size should be compared to the number of requirements changes.

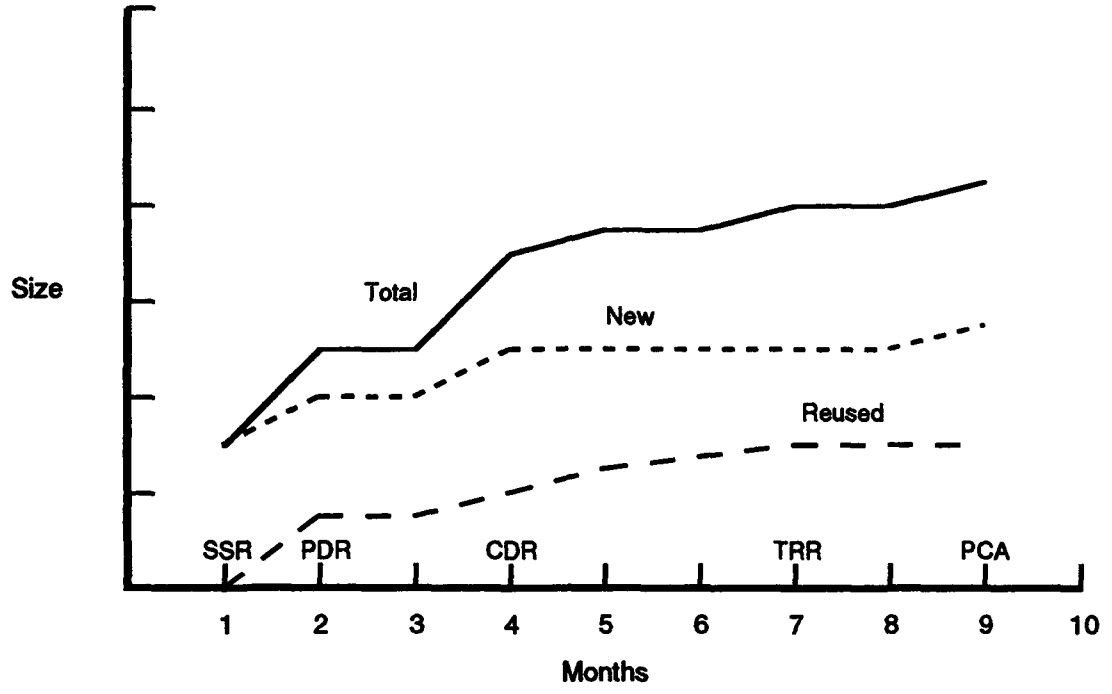


Figure 4.6.2-1. Software Size as a Function of Time

If the project is monitoring size with lines of code, the software manager should be aware of the following experiences when analyzing a figure like Figure 4.6.2-1 [Landis 90]:

- There will be periods of sharp growth in lines of code that are separated by periods of more moderate growth.
- Ten percent of the code may be produced after testing starts due to requirements changes.
- A steady growth of software size from approximately the midpoint of implementation through acceptance testing can occur due to response to trouble reports.
- Exaggerated flat spots on the curve (i.e., periods with no change) or large jumps in the curve (many changes made at the same time) indicate the need for an analysis to determine why there is no activity or why there is a sudden increase in size.
- Changes can result from a better understanding of the requirements. These changes should also be reflected in schedule and staffing. This implies that this indicator can be used in conjunction with effort and progress indicators.

Decker mentions that projects are not likely to deliver fewer lines of code than initially estimated unless a major descoping of the project occurs [Decker 91].

Figure 4.6.2-2 plots the planned and actual size of the software over time. Major deviations of the actual size from the planned size can result from the following:

- An unrealistic original estimate
- Instability in the requirements or design
- Problems in understanding the requirements
- An inexperienced team that cannot achieve the planned productivity rate
- A very experienced team that is working at higher than planned productivity rates
- An unachievable productivity rate

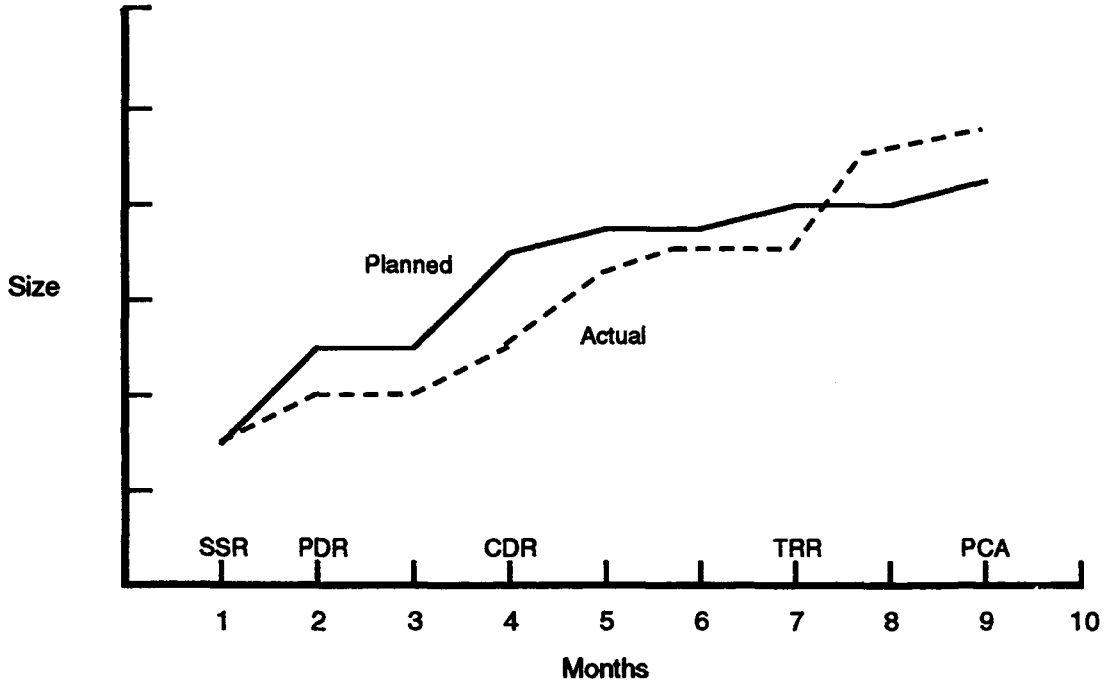


Figure 4.6.2-2. Planned and Actual Size Over Time

Figure 4.6.2-3 is an example that compares the number of software units planned for implementation in each software release and the number actually implemented. The figure is similar to that of Figure 4.6.1-4 and has the same interpretation. The only difference between the two figures is the replacement of requirements by software units. Figure 4.6.1-4 gives the project software manager an indication of how well the project is satisfying the requirements. Figure 4.6.2-3 provides similar information but in terms of software size.

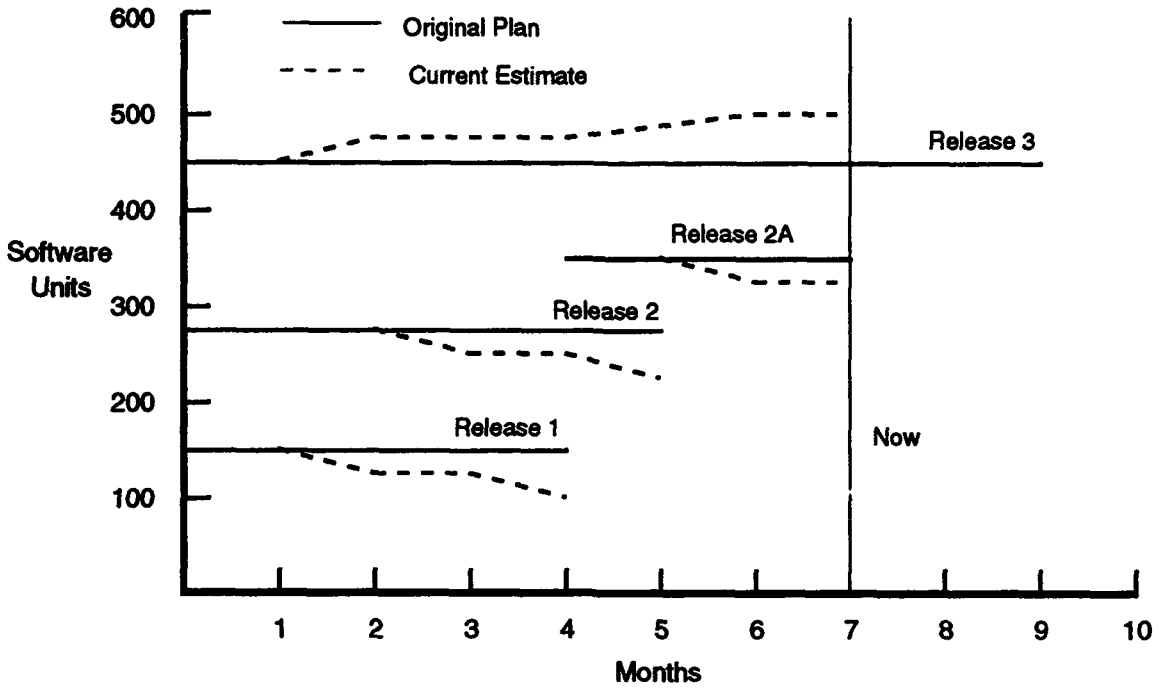


Figure 4.6.2-3. Computer Software Unit Release Content

Figures 4.6.2-1 through 4.6.2-3 show the data for the entire project. This should also be done for each CSCI.

Sources

[AFSC 86], [Decker 91], [Landis 90], [Pfleeger 89], and [Schultz 88] all discuss tracking software size.

4.7. Computer Resource Utilization

All software must work within the constraints of available system resources. For some software, the constraints significantly affect the design, implementation, and test. For those cases it is important to monitor computer resource utilization. Computer resource utilization indicators show if software is using the planned amount of system resources. They can also be used to replan, reestimate, and guide resource acquisition.

Objective of the Computer Resource Utilization Indicators

To monitor computer resource utilization to ensure compliance with requirements and to track convergence/divergence to plans and estimates.

Indicators

Trends in the deviation between planned or estimated and actual utilization

Key Process Area Goals Addressed

Software Project Planning:

- A plan is developed that appropriately and realistically covers the software activities and commitments.
- All affected groups and individuals understand the software estimates and plans and commit to support them.
- The software estimates and plans are documented for use in tracking the software activities and commitments.

Software Project Tracking and Oversight:

- Actual results and performance of the software project are tracked against documented and approved plans.
- Corrective actions are taken when the actual results and performance of the software project deviate significantly from the plans.
- Changes to software commitments are understood and agreed to by all affected groups and individuals.

Software Subcontract Management:

- The software standards, procedures, and product requirements for the subcontractor comply with the prime contractor's commitments.
- The prime contractor tracks the subcontractor's actual results and performance against the commitments.

Life-Cycle Stages

Computer resource utilization is planned during the requirements activity and reviewed during the design activity. Resources are monitored from the start of the implementation activity to the end of the life cycle.

Users

The project software manager reviews consolidated graphs of the significant or high-risk resources. Lower level managers review detailed graphs of resources for components under their control (e.g., memory used by a computer software configuration item [CSCI]).

Users' Questions

- Are the actual values of the computer resources within the allowable limits?
- Do the trends of the actuals indicate that the computer resources will remain within the allowable limits?
- Should more computer resources be acquired?

Input

System or software requirements and design that contain specifications or estimations of the maximum allowable[2] resource utilization for:

- Computer memory
- I/O throughput
- I/O channels

Interpretation

The software managers monitor the utilization level of each resource over time. Figure 4.7-1 shows a typical plot of the planned and actual utilization of memory while Figure 4.7-2 shows the actual memory used to-date expressed as a percentage of that allowed. Both figures show the maximum allowable utilization for that resource. The maximum allowable utilization may be set by physical constraints, the contract, or by modeling the software system. For example, a development plan may call for a 50 percent memory reserve for future expansion. Thus, the remaining fifty percent would be the one hundred percent allowable utilization. This number is determined by physical constraints, contract, or by modeling the software system.

For each figure, the manager compares the actual computer resource utilization with the maximum allowed. If the actual utilization is above the maximum allowed, corrective

---

[2] The maximum allowable utilization may be set by physical constraints, the contract, or by modeling the software system. For example, a development plan may call for a fifty percent memory reserve for future expansion. Thus, the fifty percent level would be the one hundred percent allowable utilization.

action is required. When examining the trend of the actual utilization line, the manager looks for sudden changes in the slope of the line. Such changes may indicate a problem. When analyzing Figure 4.7-1, the manager compares the deviation between the planned and actual lines. When the deviation exceeds a predefined limit, the manager needs to determine why and to take appropriate corrective action. As long as the deviations are small, the manager may assume that they represent no cause for concern. If, however, the deviation continues to grow during each reporting period, the manager may wish to determine the cause prior to the times when deviation exceeds the predefined limit.

Figures 4.7-1 and Figure 4.7-2 do not specify whether they are for a particular CSCI or for an entire computer system. The managers determine whether the plots will be prepared for the host and/or target machines, the software system as a whole, or for each computer software configuration item.

Other computer resources, such as database capacity or performance (e.g., response time), should be monitored where applicable. The project manager determines which computer resources will be tracked based on the nature of the project. The monitored computer resources may be development resources (host) or end-user resources (target).

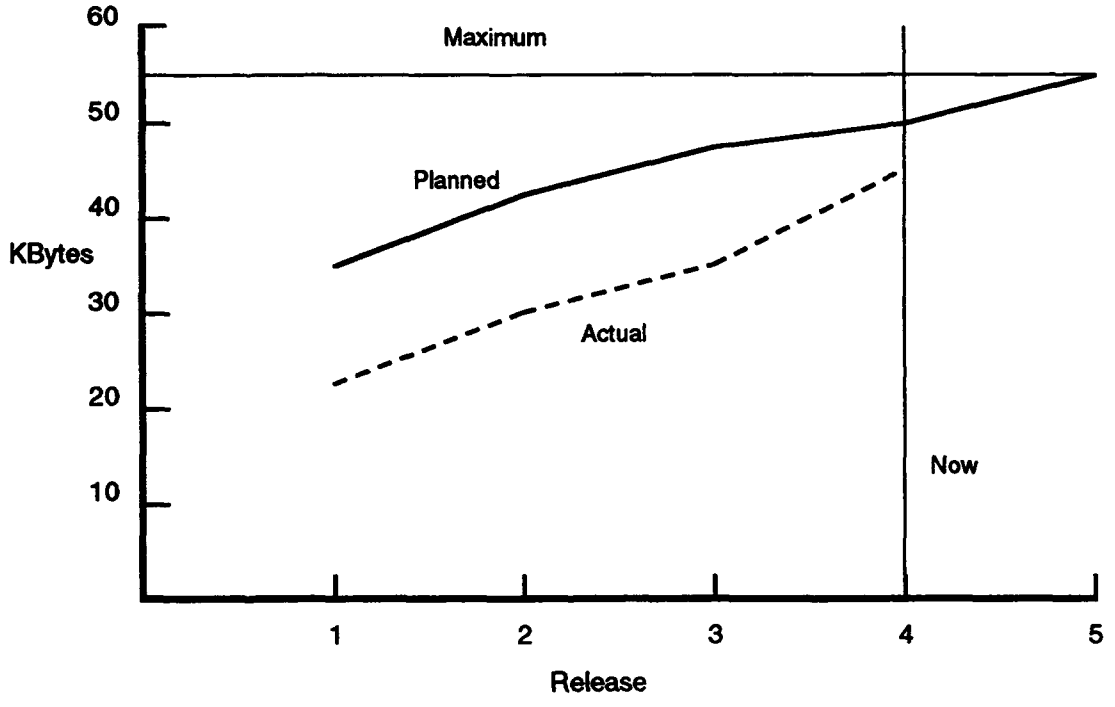


Figure 4.7-1. Planned Vs. Actual Memory Utilization

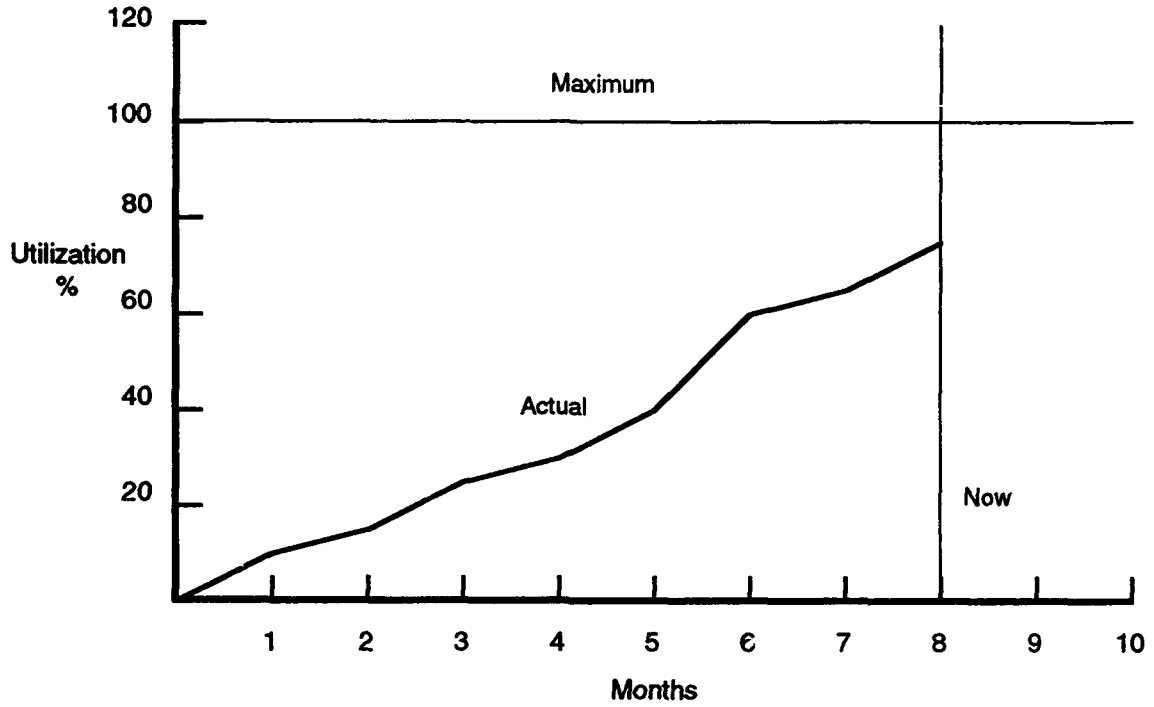


Figure 4.7-2. Memory Utilization

Sources

[AFSC 87], [Decker 91], [Landis 90], [Rozum 92], and [Schultz 88] discuss computer resource utilization.

5. The Defined Level—Maturity Level 3

This chapter summarizes the characteristics of an organization with a defined process and discusses the indicators that are appropriate for the Defined Level.

5.1. Characteristics of a Defined-Level Organization

An organization with a defined process is characterized as one with a standard process for developing and maintaining software. The process is documented and integrates both the software engineering and software management process into a coherent whole. Emphasis has shifted from project issues to organizational issues. A software engineering process group (SEPG) exists and facilitates process definition and process improvement. An organization-wide training program also exists to train the staff and managers in the skills required to carry out their tasks.

Since basic project management processes are now in place, the organization has the opportunity to concentrate on its software development processes. Managers will continue to use the indicators discussed in Chapter 4, but measures can now be made to determine the effectiveness of the organization's overall software development processes and of some detailed processes, such as peer review and training.

More historical data are available to a Defined-Level organization since it was collecting data at the Repeatable Level. An organization can use these historical data to define the normal range for measured items or by placing upper and lower bounds on these items. This allows the project manager to compare the project to the "norm." If an item is out-of-bounds, the manager has more reliable data available to determine whether the item is out-of-bounds due to a breakdown in the process or to an inherent characteristic of the item. The project manager cannot routinely perform such an analysis at the Repeatable Level since the process may not have been well-defined or even stable. However, at the Defined Level, the project manager has the advantage of more mature and defined processes. In Section 6 of their work, Landis et al have numerous illustrations of how a project manager can use ranges [Landis 90].

Indicators appropriate for a Defined-Level organization are progress, effort, cost, quality, stability, computer resource utilization, and training.

5.2. Progress

The progress indicators for the Defined Level are essentially the same as those at the Repeatable Level: Gantt charts and actual-versus-planned-completion charts (see Section 4.2). The main difference is the addition of "normal" ranges around the planned completion lines. The organization establishes the normal range for each activity by selecting some percentage around the planned completion line. This number is derived

from data reported by previous projects. The software managers then monitor the actual completions and take necessary corrective action when progress is not maintained at the planned rate.

Objective of the Progress Indicators

To provide software managers with information on the progress of the project and to monitor progress with respect to the defined ranges within each life-cycle development activity.

Indicators

From the Repeatable Level:

- The trend in the rate of progress

For the Defined Level:

- The deviation of the actual project progress outside the normal range

Key Process Area Goals Addressed

Organization Process Definition:

- A standard software process for the organization is defined and maintained as a basis for stabilizing, analyzing, and improving the performance of software projects.

Training Program:

- The staff and managers effectively use, or are prepared to use, the capabilities and features of the existing and planned work environment.

Integrated Software Management:

- The planning and managing of each software project is based on the organization's standard software process.
- Technical and management data from past and current projects are available and used to effectively and efficiently estimate, plan, track, and replan the software projects.

Software Product Engineering:

- State-of-the-practice software engineering tools and methods are used, as appropriate, to build and maintain the software system.

Intergroup Coordination:

- The project groups are appropriately involved in intergroup activities and in identifying, tracking, and addressing intergroup issues.

Life-Cycle Stages: All

Users: All levels of project software management

Users' Questions

- Are there process activities that are routinely completed late or early?
- Are the schedules of activities for the project consistent with each other and with the software requirements and the software development plan?
- Is the software subcontractor's planned schedule consistent with the software requirements and the software development plan?
- Are the actual results and performance of the software project tracked against the software development plan?
- Are the actual results and performance of the software subcontractor tracked against the software development plan?
- Is the actual performance of the project outside the range of planned completions thereby indicating schedule slippage and the necessity for a replanning effort?
- Is the actual performance of the subcontractor outside the range of planned completions thereby indicating schedule slippage and the necessity for a replanning effort?

Input

From the Repeatable Level:

- Project software development plan
- The planned and actual start and completion dates for the software development activities
- A count of tasks or task outputs planned and actually produced in the current reporting period. Task outputs include:
  - Requirements analyzed
  - Software requirements documented
  - Test cases written
  - Design elements (e.g., design diagrams) completed
  - Units designed, coded, unit tested, and integrated
  - Documents completed
  - Test cases executed
- Planned software size and the size actually completed and recorded during each reporting period

For the Defined Level:

- The normal range of deviation for each task
- A count of activities or items planned and actually accomplished in the current reporting period. These include:
  - Training classes
  - Staff taught

Interpretation

Figure 5.2-1 shows a typical actual-versus-planned completion graph, in this case a graph of the number of test cases executed. This figure is identical to Figure 4.2-3 except for the addition of the range around the planned line.

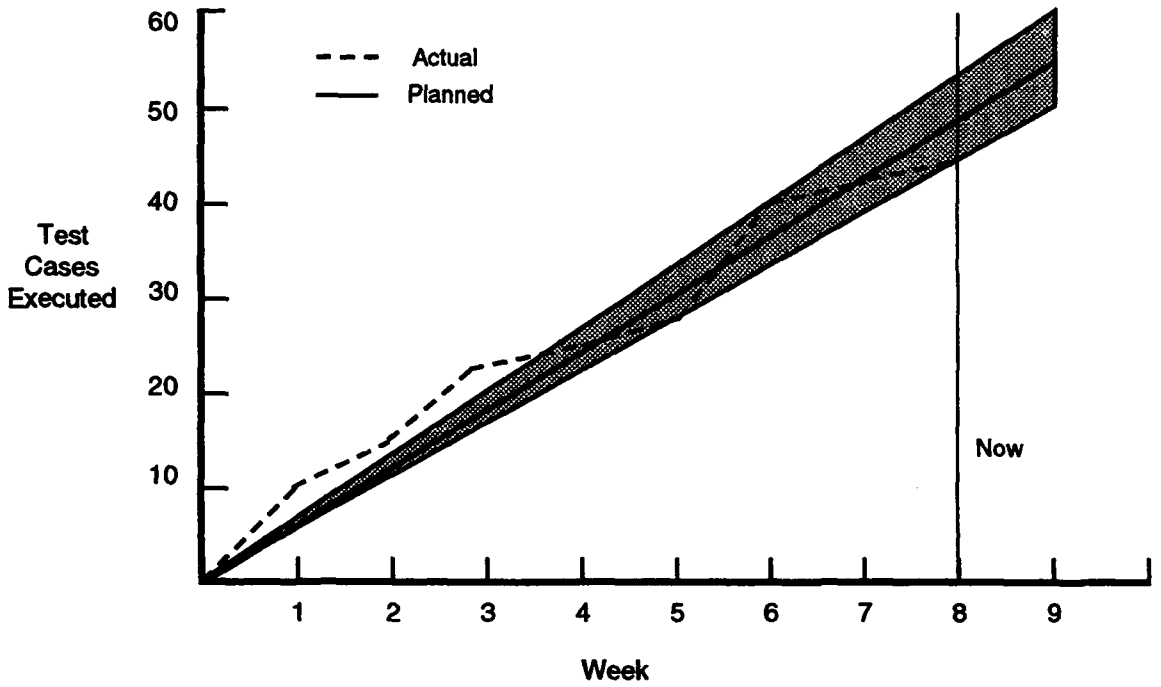


Figure 5.2-1. Actual Completions Compared with Planned Completion Range

The interpretation of the figure remains as in Section 4.2. However, the manager now has the advantage of knowing how the project tracks not only against the plan, but also against the allowable range. As long as the project stays within the allowable range, there is not much cause for concern (even though concern increases the closer the project approaches a lower limit) since the deviation may be "normal." In our example, the range is ±10 percent. The range represents a comfort zone for the project software manager. If the manager is cautious, a smaller percentage can be selected than that obtained from the historical data.

In Figure 5.2-1 the project started well; it actually performed better than planned. During week 3 and continuing to the present, the project has stayed within the ten percent range even though the project oscillated about the planned completion line. However, the trend for the last two weeks (week 6 through 8) indicates that the project is not executing the number of test cases planned. If the current trend continues through week 9, the

project will fall out of the allowable range. The manager needs to determine the reason for this trend.

The manager should also be concerned if the project exceeds the upper bound of the range, that is, the project is performing better than planned. On the surface, this appears to be good news in that the project is performing better than anticipated. However, it may indicate that other activities are not being performed or that the manager's planning and estimation processes are poor.

As in the Repeatable Level, separate graphs are prepared for the items listed in the input section depending on the life-cycle stage of the project. The number of staff taught and the number of training classes held can be tracked to provide the project software manager with information on the progress of the project training.

The use of Gantt charts is the same as for the Repeatable Level.

At the Defined Level, the project software manager analyzes the dependencies of the activities listed on the Gantt chart and conducts a critical path analysis to determine which activities lie on the critical path. This is often facilitated by the preparation of a program-evaluation-and-review-technique (PERT) chart as shown in Figure 5.2-2.

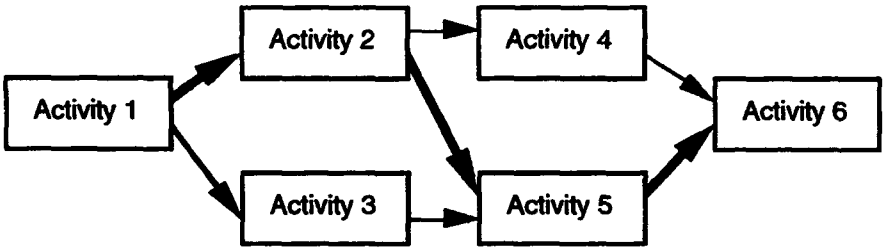


Figure 5.2-2. A Simplified PERT Chart Showing the Critical Path

This figure shows the dependencies of six activities needed to complete the project. The path defined by activities 1-2-5-6 is known as the critical path and denotes the minimum time in which the project can be completed. A slippage in the completion of any one of these activities results in a corresponding slippage in the completion of the project. During project monitoring, the project manager pays particular attention to the activities on the critical path.

Sources

From the Repeatable Level:

[AFSC 86] discusses planned and actual completions graphs at the computer software configuration item (CSCI) level.

[Decker 91] lists requirements diagrams; function specifications; design diagrams; test cases; units designed, coded, and tested; modules tested; and computer software components tested as items tracked on planned and actual completions graphs.

[Grady 87] states that calendar measures are part of the Hewlett-Packard metrics program.

[Landis 90] discusses planned and actual completions graphs for units coded, read, and tested.

[Rozum 92] has a discussion on Gantt charts in their milestone performance metric and a discussion of planned and actuals in their development progress metric.

[Schultz 88] discusses software requirements documented in his design progress metric and the number of computer software units (CSU) designed, coded, tested, and integrated in his CSU development progress metric. He also discusses the planned and actual completions of CSCIs integrated in his test progress metric.

[STEP 91] discusses a schedule metric upon which Figure 4.2-2 is based and a development progress metric.

For the Defined Level:

[Lockyer 84] discusses Gantt charts and the critical path method.

5.3. Effort

The effort indicators at the Defined Level are the same as those at the Repeatable Level. The main difference lies in the granularity of the measures, that is, effort data can be taken to a lower-level of detail. At the Defined Level, the processes are well-defined so that the managers can collect and analyze data for the lower-level activities as well as for the main activities. The staff has an understanding of the processes and can accurately record their effort at the process and subprocess level. For example, the software engineering group can record the effort expended in analyzing proposed requirement change requests and for getting agreement on the disposition of the change requests. The finer granularity, combined with data on the skill level of the staff performing the activity, allows the manager to begin to understand the staff profiles over the life of the project and to think about the suitability of the skill levels used to perform the activities.

5.4. Cost

At the Repeatable Level, managers monitored actual costs against the budgeted costs. At the Defined Level, managers can use historical data from other similar projects that allow them not only to track actual project costs against the budgeted costs, but also to compare of the performance of the project with previous, similar projects. This frame of reference is in the form of a range centered on the planned costs. Any variation from the plan that is still within this range is considered normal or low risk. Variations outside this normal range require analysis by the manager.

Also at the Defined Level, the defined processes allow the manager the opportunity to track costs to a lower-level of detail, that is, the cost of a particular activity or sub-activity can be determined. For example, the manager can track the costs of process definition and improvement activities. The manager can then use this knowledge in an analysis of the process or a change to the process.

Objective of the Cost Indicators

To track actual costs against the project plan and to predict future project costs.

Indicators

- Performance of the actual cost of work performed against budgeted cost for work scheduled
- Performance of the budgeted cost for work performed against the budgeted cost for work scheduled
- Trends in the cost variance
- Trends in the schedule variance
- Trends in the cost and schedule performance indices

Key Process Area Goals Addressed

Integrated Software Management:

- The planning and managing of each software project is based on the organization's standard software process.
- Technical and management data from past and current projects are available and used to effectively and efficiently estimate, plan, track, and replan the software projects.

Life-Cycle Stages: All

Users

All levels of management. The managers cited here are not restricted to software development managers but include managers of support groups, for example, software quality assurance, software configuration management, training, and so forth.

Users' Questions

- Are the actual costs and performance of the software project tracking against the plan?
- Are the actual costs within the range established for the software project?
- Are the corrective actions reducing the cost and variances?

Input

- Budgeted cost for work scheduled (BCWS)
- Budgeted cost for work performed (BCWP)
- Actual cost of work performed (ACWP)
- Budgeted cost at completion (BCAC)

Interpretation

Figure 5.4-1 shows a typical cost/schedule status report. This figure is identical to Figure 4.4-3 except for the range around the budgeted cost for work scheduled. The organization establishes the normal range by selecting some percentage around the budgeted-cost-for-work-scheduled line. This number is derived from data reported by previous projects.

The interpretation of Figure 5.4-1 is the same as in Section 4.4. By adding the range, it is easy to tell at a glance how poorly this project is tracking against the plan. The software manager needs to determine the causes of the cost and schedule variances and to determine appropriate corrective action.

In Figure 5.4-1 the cost performance to-date is favorable, since the ACWP tracks below BCWP (cost variance is positive). However, the rate of expenditures increased, as evidenced in the increased slope of ACWP during the past two months. The BCWP line stayed relatively flat. These two lines indicate that the project is spending money, but not accomplishing the work that has been scheduled. Accordingly, the favorable cost variance that has accrued in previous months is eroding. There is also an unfavorable schedule situation (schedule variance is negative). The BCWP continues to track below the BCWS line, indicating that this project is behind schedule. However, since BCWP and BCWS are converging slowly, the amount of slippage is decreasing.

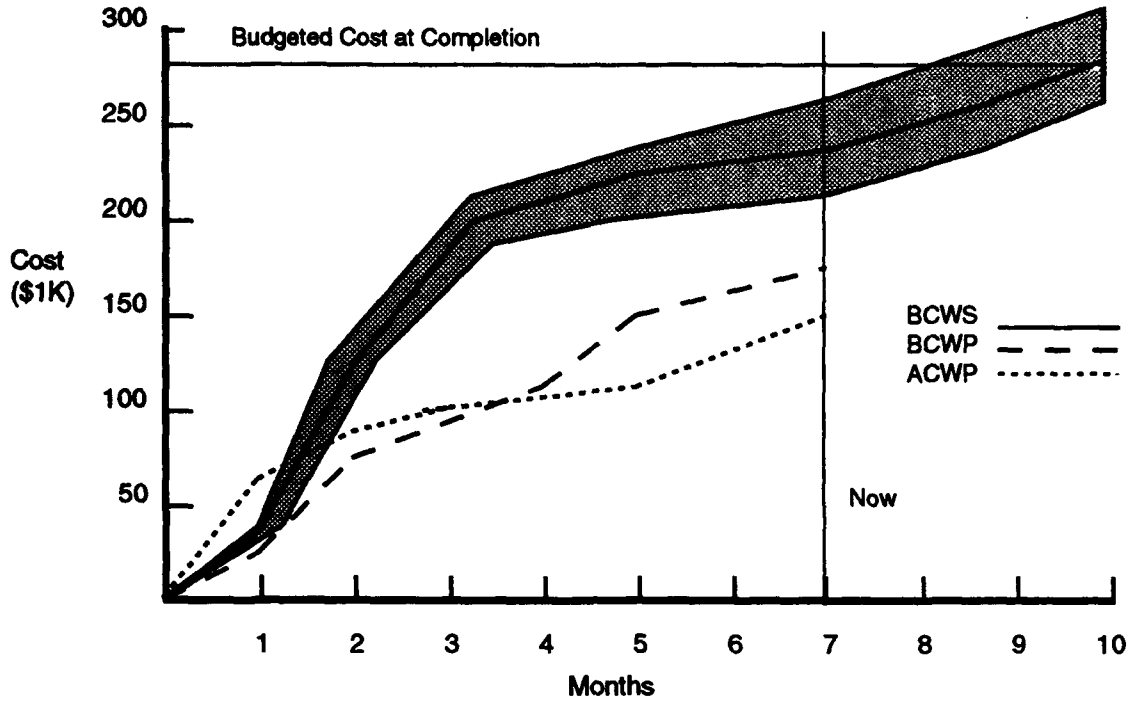


Figure 5.4-1. Cost/Schedule Status Report Showing Allowable Range for Budgeted Cost for Work Scheduled

Figure 5.4-2 shows the performance of the cost and schedule variances over time for a different project. The figure focuses on the variances themselves and usually illustrates more convincingly the problems or successes of the project.

Figure 5.4-2 shows that this project began with a negative cost variance, achieved a turn around, but again in month 6 returned to a negative slope. The steep negative slope from month 5 through month 7 indicates that the project is headed for a serious problem. In fact, the large negative schedule variance in the preceding months actually foretold the problem. In the attempt to make up for the schedule problem, the project has over spent and thus brought about a forthcoming cost overrun condition.

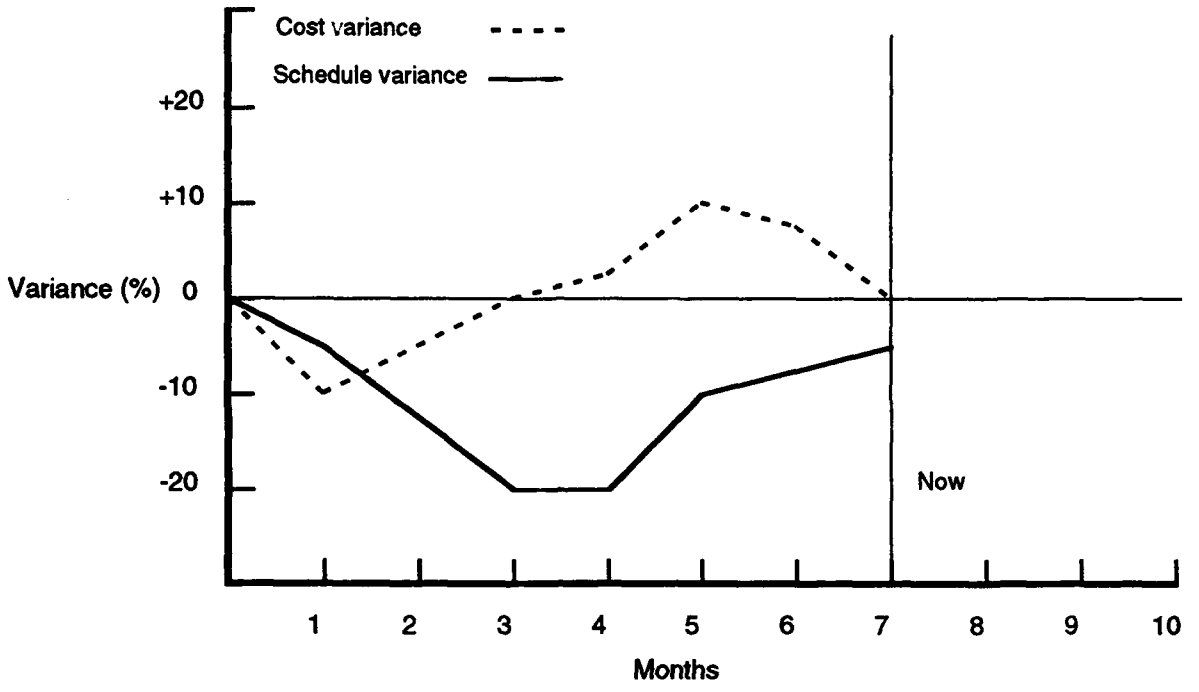


Figure 5.4-2. Variance Report

Variance is to be expected; the amount and sign of variance is important. An organization uses its historical data to determine the normal ranges of cost and schedule variances. An organization uses these ranges to establish threshold limits on variances. For example, an organization can adopt ±10 percent to signify low risk, ±10-15 percent to signify moderate risk (and the possible need for corrective action), and any variance over ±15 percent to signify a high risk that requires corrective action. The manager of the area that exceeds the threshold explains to the next level of management the cause(s) of the variance and the corrective action to be taken (if any). If variances consistently remain beyond the thresholds, the organization may wish to reexamine its estimating and scheduling processes. Likewise, if the variances consistently stay within the threshold, management may want to redefine the threshold by setting smaller tolerances.

When the variances become so large that the plan is no longer valid, the managers replan the elements of the project according to the documented procedure for the project.

Another way of analyzing the performance of the project is through cost and schedule performance indices. The cost performance index (CPI) is the ratio of the budgeted cost for work performed and the actual cost of the work performed, expressed as a percentage [CPI=(BCWP/ACWP)*100%]. The schedule performance index (SPI) is the ratio of the budgeted cost for the work performed and the budgeted cost for the work scheduled, expressed as a percentage [SPI=(BCWP/BCWS)*100%].

Figure 5.4-3 illustrates a sample behavior of the cost performance, and schedule performance indices. Both performance indices are really efficiency indicators. A cost performance index of fifty percent indicates that only half as much budgeted work was accomplished as the actual cost to achieve it. This is very poor performance, and the reason for that performance must be determined. Similarly, if half of the scheduled work, measured in dollars, had actually been performed, the schedule performance index would be fifty percent.

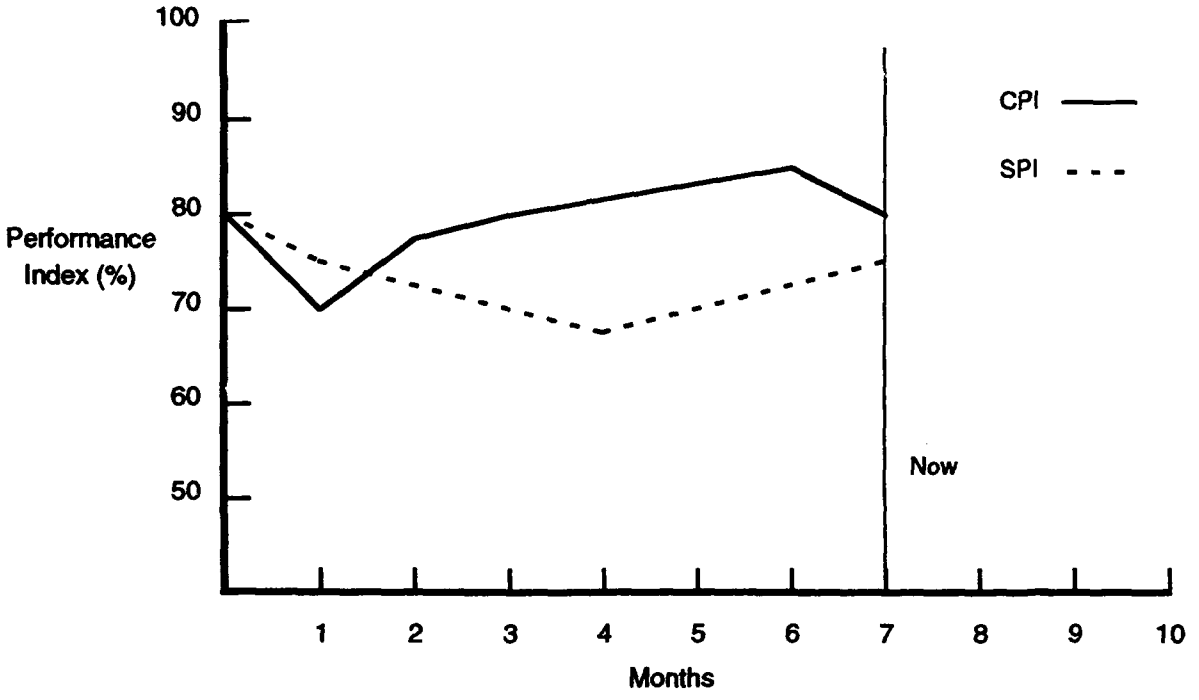


Figure 5.4-3. Example of a Performance Indices Report

Figures 5.4-2 and 5.4-3 represent typical monthly reports. They are prepared for each software manager, with the higher-level software managers receiving a consolidated report for the area(s) of their responsibility. The higher-level software managers also have access to the lower-level data and graphs.

The budgeted cost at completion (BCAC) shown in Figure 5.4-1 represents the manager's original estimate for the total cost of work. Each reporting period, the manager should develop an estimated cost at completion. This estimate is based on the actual cost of work performed to date and the manager's estimate of the cost for the remaining work. To obtain this estimate, the manager considers the project performance to date (current and cumulative variances and productivity), the impact of approved corrective action plans, and known or anticipated problems. This estimate is often based on the manager's professional judgment. A check on this judgment can be obtained

through the use of the actual costs and the performance indices. For example, an estimated cost at completion can be obtained from the following equation [AFSC 86]:

Estimated cost at completion = ACWP + [BCAC - BCWP] / [0.2(SPI) + 0.8(CPI)]

Other estimates can be obtained through the use of parametric cost models. Ideally, the organization should develop its own equation from its historical cost data. The manager compares the estimated cost at completion with that derived from the data and explains any differences. The final estimated cost at completion must be reviewed for realism and appropriateness.

Sources

[AFSC 86] has a discussion of these indicators.

[DoD 80] discusses the basics of ACWP, BCWP, and BCWS.

[DSDM 89] served as the major source of information in this section.

5.5. Quality

The Defined Level quality indicators are divided among the results of software quality assurance audits, the results of life-cycle reviews with the customer, the trouble reports written after the implementation team has released the software for testing, and the results obtained from peer reviews.

5.5.1. Software Quality Assurance Audit Results

The quality indicators from software quality assurance audits are essentially the same as those at the Repeatable Level (see Section 4.5.1). One difference is that software quality assurance personnel routinely conduct process audits as well as product audits since the organization's process is now defined. Also, the software quality assurance organization begins to collect data about the audit process itself, for example, number and types of audits.

During a process audit, the software quality assurance group audits the processes for developing and revising the project's defined software processes; managing the project's software cost, schedule, risk, and technical activities; and using and controlling the software process database for software planning and estimating.

Objective of the Software Quality Assurance Audit Results

To provide project software management with an independent evaluation of the quality of the product and the adherence of the project staff to the processes that created the product.

Indicators

- Trends in the number, type, and severity of noncompliance issues found during an audit
- Trends in the rate at which the noncompliance issues are being addressed

These indicators are the same as the Repeatable Level, but they apply as well to the process audits of the higher maturity levels.

Key Process Area Goals Addressed

Organization Process Definition:

- A standard software process for the organization is defined and maintained as a basis for stabilizing, analyzing, and improving the performance of the software projects.

Integrated Software Management:

- The planning and managing of each software project is based on the organization's standard software process.

Life-Cycle Stages: All

Users

- Mid-level managers
- Project software manager
- Senior management
- Software quality assurance

Users' Questions

From the Repeatable Level:

- Are audits conducted by an independent software quality assurance group for each step of the software development process?
- Are standards and procedures applied on the software project?
- Are project personnel applying the standards and procedures correctly?
- Are project personnel following the standard processes (where defined)?
- Are independent audits conducted for the software subcontractor to ensure compliance with the software development plan?
- Are the noncompliance issues being addressed in an appropriate manner?
- Are the noncompliance issues being addressed in a timely manner?

For the Defined Level:

- What types of noncompliance issues are detected during an audit?
- Is the software quality assurance organization reviewing a sufficient number of samples during its audit?

Input

From the Repeatable Level:

- Number of noncompliance issues:
  - Total
  - Open
  - Closed
- For each noncompliance issue:
  - Date opened
  - Type (e.g., a noncompliance of the product to a standard or noncompliance to a process)
  - Severity (degree of noncompliance: e.g., product cannot be delivered as is, product must be corrected by next release, process change be recommended)
  - Date closed

For the Defined Level:

- Number of audits conducted
- For each audit:
  - Type of audit, for example, product audit or process audit
  - Sample size (for each product audit)

Interpretation

Regardless of whether a product or process audit is conducted, the figures and their interpretation are the same as the Repeatable Level in Section 4.5.1. At the Defined Level, however, the project and organization can use the audit results and the indicators for peer reviews (Section 5.5.4) to evaluate inspection efficiencies. Figure 5.5.1-1 compares the number of policy, product, and process noncompliance issues detected during software quality assurance audits in a reporting period.

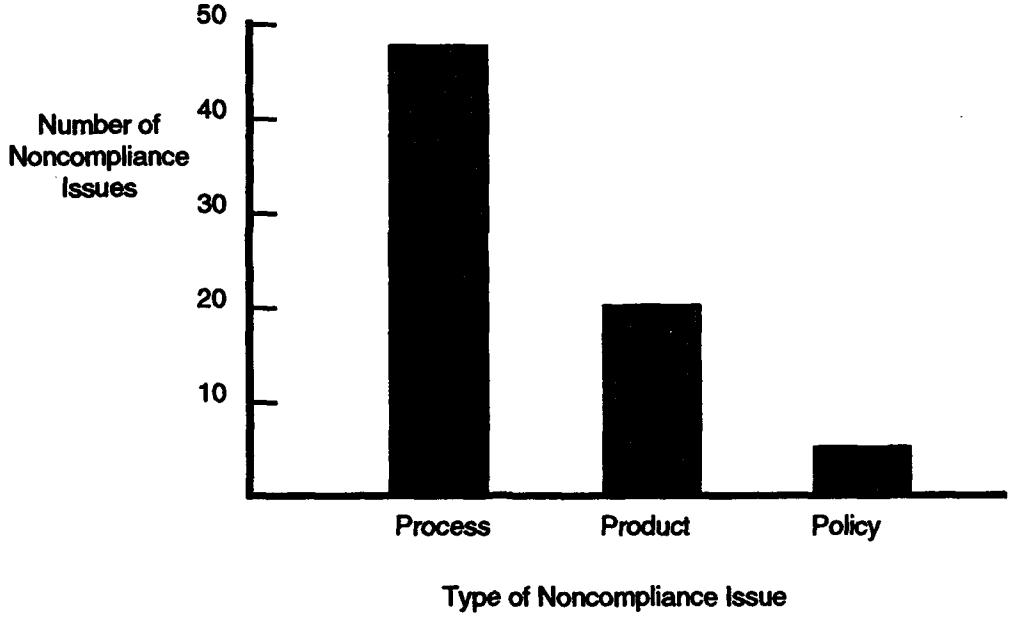


Figure 5.5.1-1. Number of Noncompliance Issues by Type

The figure shows that the peer review process is effective since few product noncompliance issues are reported. These noncompliance issues have presumably been found in peer reviews and were corrected prior to the audits and delivery. This can be verified by analyzing the peer review indicators.

The software quality assurance manager can also prepare a report on a regular basis that summarizes the software quality assurance organization's audit activities. This

report can include the number of each type of audit conducted and summary information on each audit. This information includes sample size and the percentage of the total that the sample size represents.

Sources

[Pfleeger 89] and [Florac 92] discuss the tracking of problems found in a variety of life-cycle stages. The indicators of software quality assurance audit results are an extension of that discussion.

5.5.2. Review Results

The quality indicators from reviews are the same as those at the Repeatable Level (see Section 4.5.2). Also, the software quality assurance organization begins to collect data about the review process itself.

Objective of the Review Results Indicators

To provide software project management, senior management, and the customer with the status of action items originating during a life-cycle review.

Indicators

From the Repeatable Level:

- Trends in the number, type, and priority of action items recorded during a review
- Trends in the rate at which the action items are being addressed

Key Process Area Goals Addressed

Integrated Software Management:

- The planning and managing of each software project is based on the organization's standard software process.

Life-Cycle Stages: All

Users

From the Repeatable Level:

- Project software manager
- Customer
- Senior management
- Software quality assurance
- Software engineering

Users' Questions

From the Repeatable Level:

- Are the action items being handled in a timely manner?
- How are the number and types of open action items going to impact the cost, schedule, and resources?

For the Defined Level:

- Are the process action items being handled as expeditiously as the product action items?

Input

From the Repeatable Level:

- Number of action items:
  - Total
  - Open
  - Closed
- For each action item:
  - Date opened
  - Type (e.g., documentation change, additional analysis required, resolution of a to-be-determined item, process)
  - Priority (e.g., relative ranking of importance of the action item)
  - Date closed

Interpretation

The figures and interpretations are the same as the Repeatable Level in Section 4.5.2.

At the Defined Level the organization has its processes defined and has historical data to use. One use is to predict how long an activity will take. The project software manager can use the data from previous projects and from earlier project reviews to predict how long it will take to close an action item of a given priority or type. This information can be obtained by determining the average length of time the action items of each priority or type from previous reviews on similar projects remained open. The manager can determine the performance of the project against these averages. Table 5.5.2-1 compares the performance of a project in addressing review action items against the averages obtained from historical data.

Action Items	Average Value	Project Value to Date
Product Priority		
1	4 hours	3 hours
2	4 hours	4 hours
3	1 week	1 week
4	1 month	3 weeks
All Product	9 days	7 days
All Process	3 months	3.5 months

Table 5.5.2-1. Length of Time Action Items Remain Open

The project is addressing the highest and lowest priority action items faster than the average project and is performing just as an average project with respect to the middle priority action items. The project is also addressing all product action items (all priorities combined) faster than the average, but is not performing as well with respect to the process related action items.

Sources

From the Repeatable Level:

[AFSC 86] discusses tracking action items in its requirements definition and stability indicators.

[Florac 92] discusses tracking action items in their general discussion of a problem management system.

[Rozum 92] discusses tracking action items in their software defects metric.

[Schultz 88] discusses tracking action items in his software volatility metric.

5.5.3. Trouble Reports

The quality indicators from trouble reports include the indicators from the Repeatable Level. At the Defined Level, the organization begins to gather and analyze the information on its trouble reporting process and on the development processes.

Objective of the Trouble Reports Indicators

To provide software managers with an insight into the quality of the product, the software reliability, and the effectiveness of testing and to provide the software engineering process group with information on the development processes.

Indicators

From the Repeatable Level:

- Trends in the following:
  - Number, type, and severity of the trouble reports
  - Trouble report density, that is, the number of trouble reports per unit size
  - Rate at which trouble reports are being addressed
- Relationship between the number of trouble reports and the number of test cases passed

For the Defined Level:

- Trends in the following:
  - Rate at which trouble reports are being written
  - Number of defects in each software component

Key Process Area Goals Addressed

Software Product Engineering:

- The software engineering activities are well-defined, integrated, and used consistently to produce a software system.
- State-of-the-practice software engineering tools and methods are used, as appropriate, to build and maintain the software system.

Life-Cycle Stages

Integration and acceptance test

Users

From the Repeatable Level:

- Project software manager
- Software quality assurance personnel
- Software testing manager
- First-line and mid-level software development managers

For the Defined Level:

- Software engineering process group

Users' Questions

From the Repeatable Level:

- Does the quality of the product indicate that it is ready for release to the customer?
- Will undetected or unresolved problems in the product lead to more problems in the next life-cycle stage?
- Does the number of trouble reports indicate that the software product should be reworked before proceeding to the next life-cycle stage?
- Is the testing activity complete?
- Are project personnel addressing the trouble reports in a timely manner?

For the Defined Level:

- Do the types of defects suggest areas for process changes?
- How does this project compare to others with regard to the number and types of defects discovered?
- Which components tend to be error prone?

Input

- Number of trouble reports:
  - Total
  - Open
  - Closed
- Number of test cases:
  - Scheduled
  - Passed
- Product size
- For each trouble report:
  - Date opened
  - Date closed

- Date trouble report evaluated
- Type
- Severity
- Type (category) of defect
- Severity of defect
- Product identification
- Trouble report identification
- Source of problem
- Cause of defect
- Life-cycle stage in which trouble report is written
- Activity in which defect was introduced
- Units affected by defect

Note that some of the information requested for each trouble report can be provided only after the problem has been analyzed and the defect fixed.

Interpretation

The plots used at the Repeatable Level are also applicable at the Defined Level. At the Defined Level, the figures are more meaningful if prepared for each computer software configuration item (CSCI) instead of for the project as a whole. Also, at the Defined Level, the project can use historical data to compare itself against similar projects and to predict the number of trouble reports it can expect.

Figure 5.5.3-1 compares the total number of trouble reports written against the range of the average total number of trouble reports written per week obtained from historical data. The project software manager needs to determine why at week 7 this CSCI started to exceed the norm. Did the test team postpone the start of the more complex tests? Has the quality of the CSCI suddenly decreased? Have the early tests not been sufficient to detect these defects? How does this CSCI compare to the other CSCIs on the project? In general, if the number of trouble reports exceeds the upper threshold, possible causes are unreliable software, misinterpreted requirements, or extremely complex software. If the number of trouble reports is less than the lower threshold, possible causes are reliable software or inadequate testing.

Figure 5.5.3-2 is an example of what an organization can do with historical data. This figure (adapted from [Landis 90]) shows the number of errors detected per thousand source lines of code (KSLOC) for each life-cycle stage for the project and the range observed from the historical data. Landis et al report that their data in the Software Engineering Laboratory supports a "halving" model in which the rate is cut by fifty percent at each stage [Landis 90]. Typically, their projects have about four errors per thousand SLOC during build/release testing, two errors per thousand during system testing, and one error per thousand during acceptance testing.

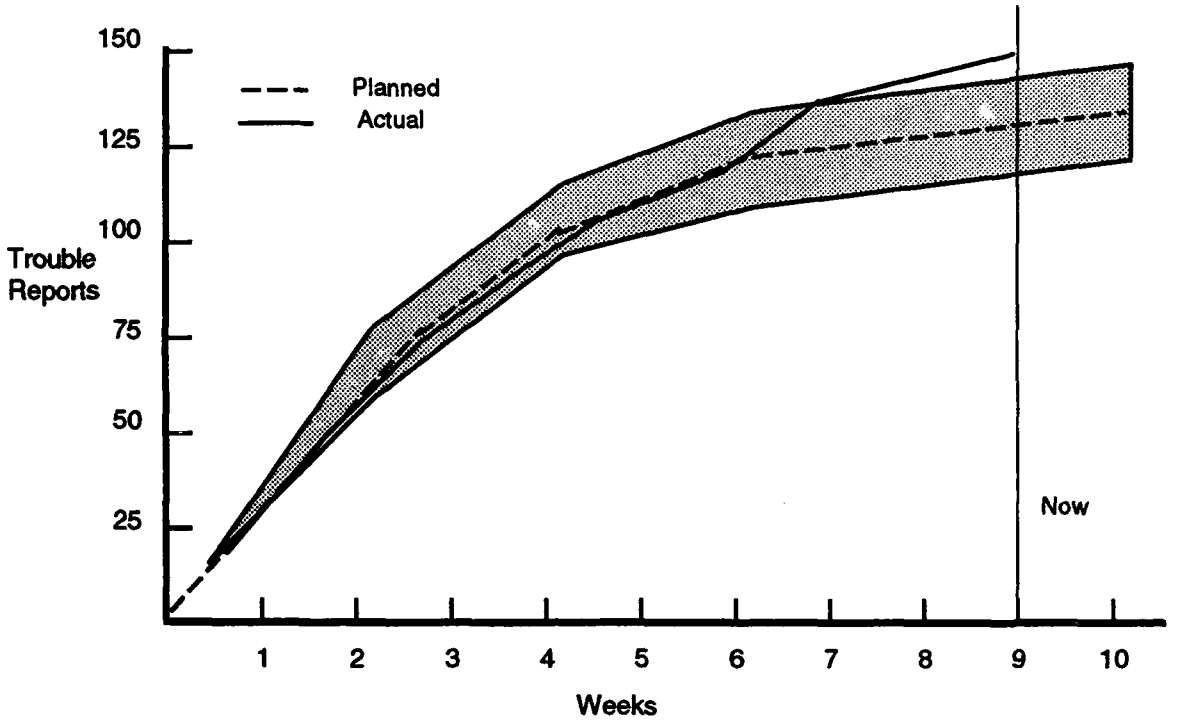


Figure 5.5.3-1. Number of Trouble Reports Compared to Range Determined from Historical Data for Similar Projects

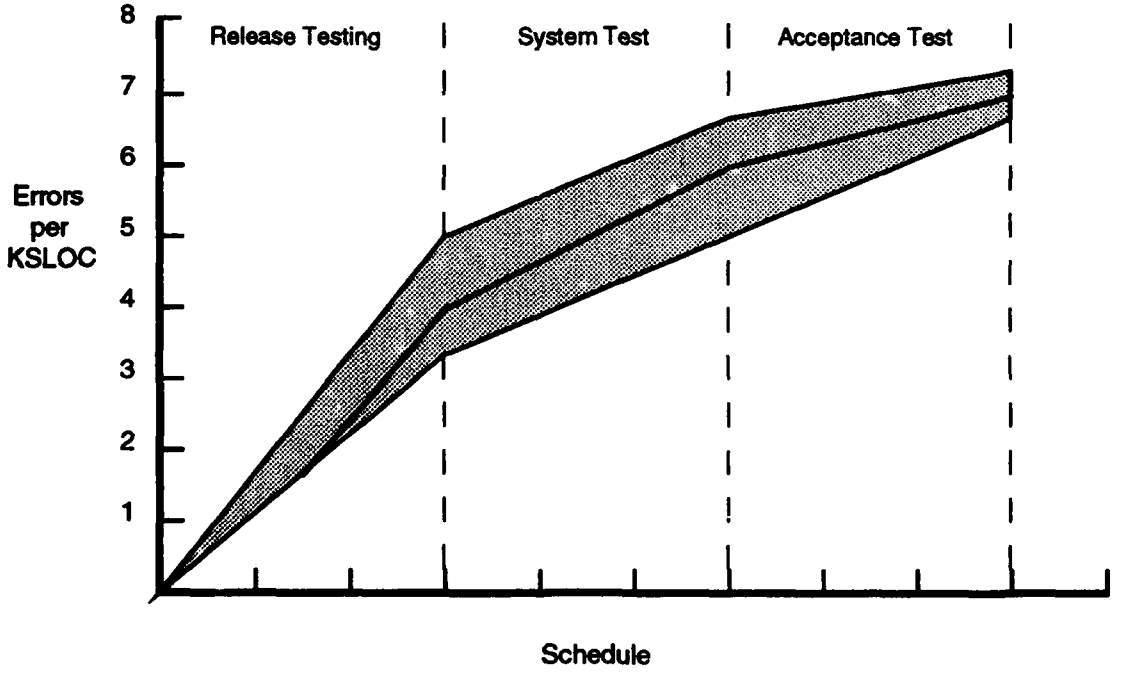


Figure 5.5.3-2. Software Engineering Laboratory Error-Rate Model

In addition to plotting the total number of trouble reports written for each CSCI, the project software manager also looks at the plots of the number of open and closed trouble reports. One way of looking at these is shown in Figure 4.5.3-1. Figure 5.5.3-3 shows an alternative way of tracking open trouble reports. This figure, prepared for each reporting period, shows how long trouble reports of a given severity remain open. Such a figure is most useful for higher levels because these are the problems that are addressed first. Lower severity problems are usually fixed when there are several to repair or on a time-available basis until the number of problems at that level becomes high. The figure indicates how rapidly the software development team is addressing the problems at this severity. If length of time becomes unreasonably long, the manager needs to know and understand the reason(s).

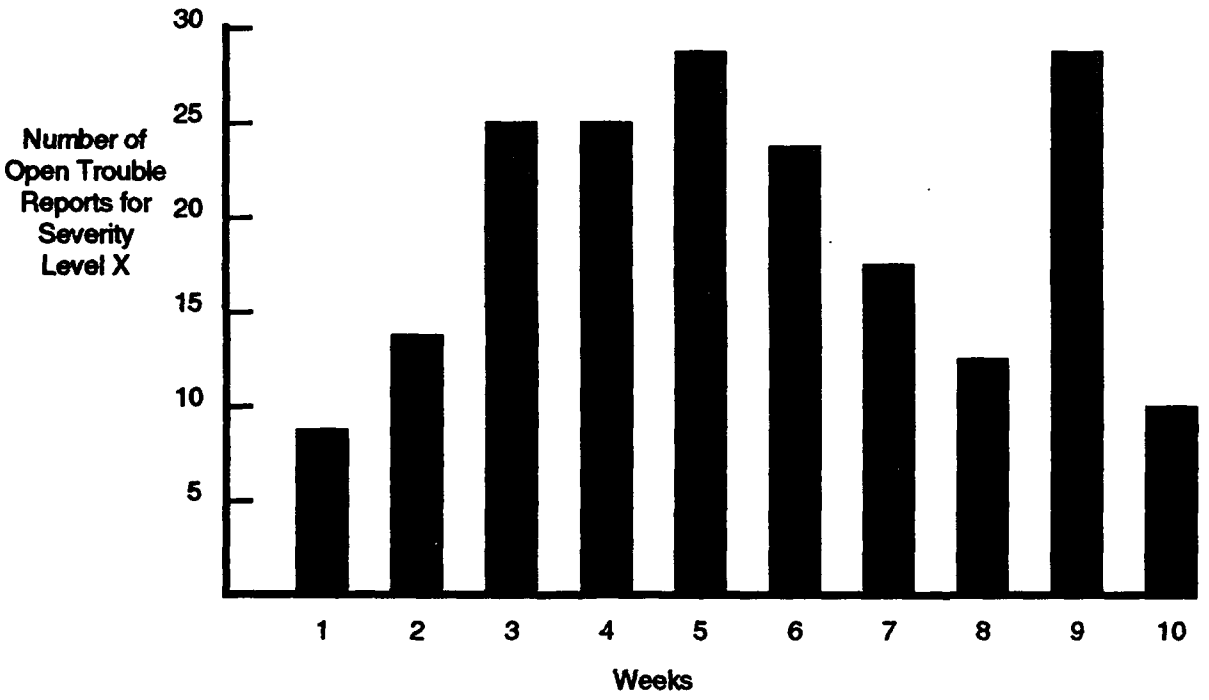


Figure 5.5.3-3. Length of Time that Severity Level X Trouble Reports Are Open

Figure 5.5.3-4 shows the number of trouble reports written for each software work product. In this example, CSCIs are used as the work product, but the figure can also use modules, units, or computer software components as the work products. This figure is intentionally left ambiguous with regard to frequency of reporting. If the software products are being tested in parallel, then the comparison on a weekly, biweekly, or monthly basis is valid. If, however, the work products are tested sequentially, the figure is meaningful only at the end of the testing activity, that is, when the data for all products

are available. Also, to allow for varying product sizes, the number of trouble reports written should be normalized, for example, the number written per thousand lines of code. Lastly, the manager may want to draw the figure by severity level, rather than using all severity levels to eliminate any dominating effect which lower severity level trouble reports may have. Regardless how the specific figure is drawn, it allows the project software manager to do the following:

- Compare trouble report data for multiple related work products
- Identify those work products that may require further analysis to determine possible process or product problems resulting in the higher number of trouble reports
- Identify work products on which few trouble reports were issued. These products may also warrant some analysis to determine what is being done right so the same processes can be used for other similar products.
- Identify products that are of lower quality

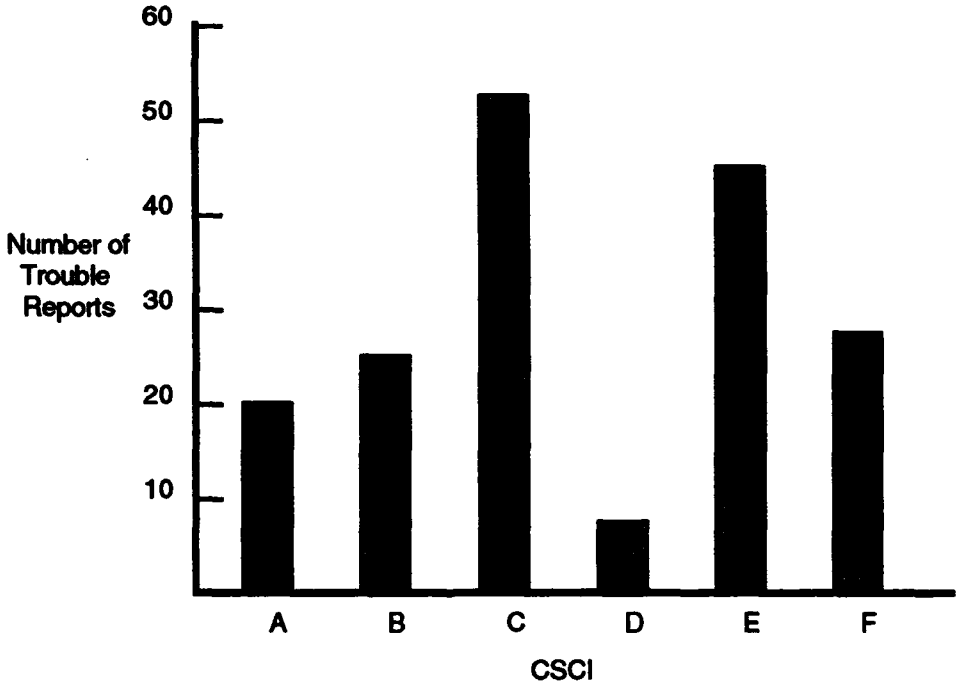


Figure 5.5.3-4. Total Number of Trouble Reports per Computer Software Configuration Item

Sources

From the Repeatable Level:

[AFSC 87], [Buckley 90], [Card 90], [Decker 91], [Grady 87], [IEEE 1061], [Landis 90], [Pfleeger 89], and [Rozum 92] discuss trouble reports.

[Florac 92] has a thorough discussion of trouble reports and serves as the main source of information for this section.

From the Defined Level:

[Florac 92] has a thorough discussion of problem reports.

[IEEE 1044] was used to determine the inputs for this indicator.

5.5.4. Peer Review Results

Peer reviews can be a powerful tool for the project software manager. The basic principle of the peer review process is to find defects in the intermediate software products and remove them so that the final product is of high quality. Peer reviews should begin early in the life cycle. An error discovered during the operations stage may be as much as one hundred times more expensive to repair than if it were corrected in the preliminary design stage.

Measurements taken from peer review data can provide information on the number and types of defects recorded. These data document the efficiency and efficacy of the peer reviews and can point to potential problems in the processes used to develop the product, as well as identify defects in the product.

Objective of Peer Review Results Indicators

- To provide the software managers and the software quality assurance personnel insight into the quality of the intermediate and final products.
- To provide the SEPG with insight into the peer review and development processes.

Indicators

Trends in the following:

- Number of defects detected during peer reviews
- Type and severity of defects
- Number of re-reviews required
- Rate at which defects are being addressed
- Number of defects in each software component
- Number of defects in each product type
- Defect detection efficiency

Key Process Area Goals Addressed

Peer Reviews:

- Product defects are identified and fixed early in the life cycle.
- Appropriate product improvements are identified and implemented early in the life cycle.
- The staff members become more effective through a better understanding of their work products and knowledge of errors that can be prevented.
- A rigorous group process for reviewing and evaluating product quality is established and used.

Software Product Engineering:

- The software engineering activities are well-defined, integrated, and used consistently to produce a software system.
- State-of-the-practice software engineering tools and methods are used, as appropriate, to build and maintain the software system.

Life-Cycle Stages: All

Users

- Software managers
- Software quality assurance personnel
- Software engineering process group

Users' Questions

- Does the peer review indicate that the product quality is sufficient to allow the product to proceed to the next stage?
- Does the number of re-reviews suggest that there may be product quality and/or process problems?
- Do the types of defects detected during peer reviews suggest areas for process change?
- Do the peer reviews effectively detect defects?

Input

- For each peer review:
  - Type of review (e.g., requirements review, design review, code review)
  - Number of items reviewed
  - Action items open
  - Action items closed
  - Identification of product reviewed
  - Product size
  - Preparation lead time
  - Preparation time for each reviewer
  - Length of time of review
  - Size of review team
  - Experience of review team
  - Structure of review team
  - Number of defects detected

- For each defect:
  - Severity
  - Type
  - Rework effort
  - Life-cycle stage in which it was introduced into product
  - Number of units affected by defect
  - Number of units containing defect
- Number of peer reviews
- Number of re-reviews

Note: Some of the inputs may not be used until higher maturity levels.

Interpretation

Defects detected during testing are not discussed in this section. They are discussed in the trouble reports indicator Sections 4.5.3 and 5.5.3. The defects discussed here are those detected during peer reviews. However, techniques and graphs discussed in this section can also be used for the defects detected through testing and recorded on the trouble reports.

Similarly, action items are not treated in this section. As noted in Section 4.5.2, the action items detected in peer reviews need to be tracked to closure, but the time scales usually involved with the peer review process do not allow for the formalism proposed in Section 4.5.2. A manager may choose to treat peer review action items in the same fashion as in Section 4.5.2.

Figure 5.5.4-1 shows the number of defects open and closed and the total number of defects versus time. This graph is similar to Figure 4.5.3-1 and has similar interpretation.

Figure 5.5.4-2 shows an alternate view of the data. In this figure, the percentage of defects corrected against the reported total is plotted against time. This provides a measure of the rate at which the development team is responding to the defect reports. In the figure, the development team exhibited a tendency not to actively pursue the correction of defects. After week 3, the team's performance with respect to the defects improved and remained fairly constant around seventy-five percent until week 7, after which the team was very responsive to the defects. If the team had not changed its performance, a backlog of defects would have occurred. When analyzing Figure 5.5.4-2, the managers should be cognizant of the size of the changes. If the majority of the defects addressed by the team require only minor changes to the product, only a small amount of effort may be needed to fix them. This can falsely indicate that the development personnel are addressing the defects in a timely manner when, in reality, major rework may be required to close a defect.

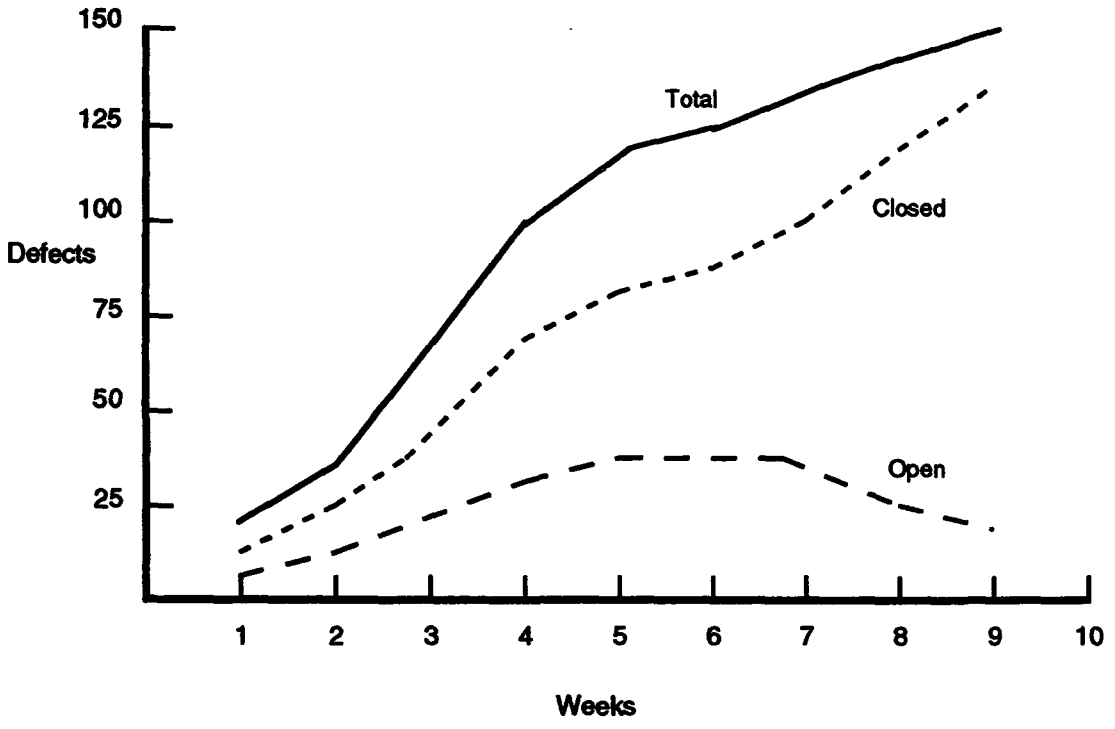


Figure 5.5.4-1. Number of Total, Open, and Closed Defects

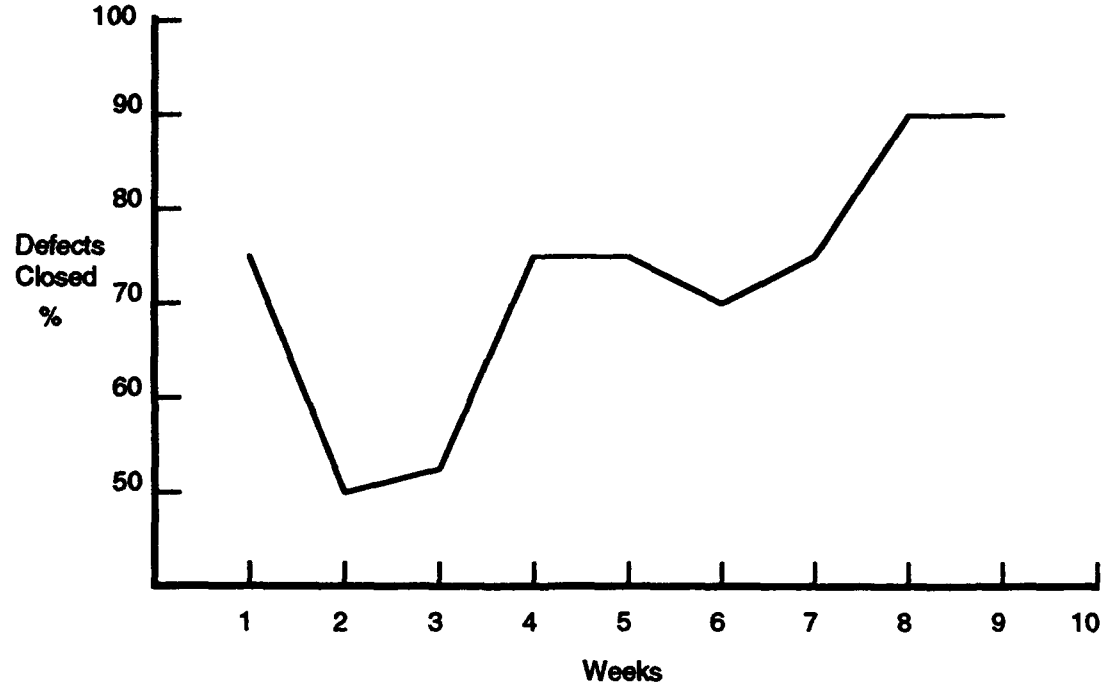


Figure 5.5.4-2. Percentage of Total Defects Closed

Upon completion of a particular life-cycle activity, a Pareto analysis can be performed on the types of defects discovered during that activity. Figure 5.5.4-3 shows an example of the distribution of several possible defect categories for unit design reviews. The figure can be used to determine which categories are the major source of defects in a particular type of review. In this example, failure to be compliant with standards is the dominant defect category. Interface errors also represent a major defect category. The figure indicates that software quality assurance personnel could teach a refresher course on the unit design standards prior to the next unit design activity. In addition, the SEPG may wish to review the design process to determine whether the process can be changed in a manner to lower the relatively high value of interface defects. However, the defects may be the result of carelessness or sloppiness on the part of the designers.

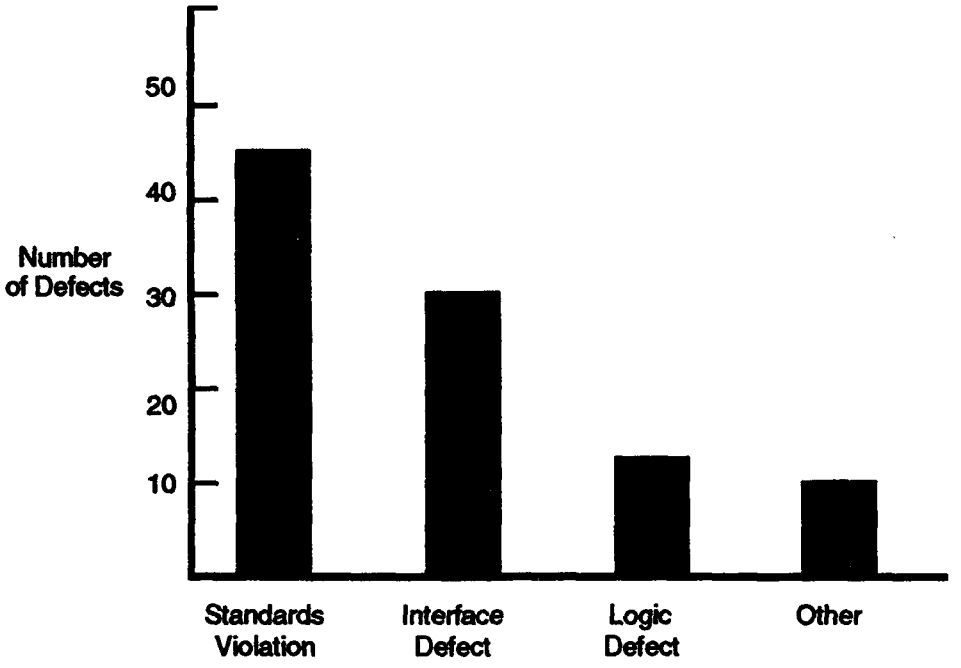


Figure 5.5.4-3. Distribution of Defect Categories for Unit Design Reviews

Figure 5.5.4-4 shows the number of defects discovered in each software product (CSCI). The figure can be drawn at the module or computer software component level. The manager or software quality assurance personnel use the figure:

- To compare defect data for multiple related work products.
- As a predictor of which CSCI or software component is more likely to generate more trouble reports, require more testing, or require more maintenance effort after delivery.

- To identify those work products which may require further analysis to identify possible process or product problems causing a higher number of defects.
- To determine a work product for which few defects were issued. This product may warrant some analysis to determine why it had fewer defects than the other products. If a different process was used, perhaps it can be applied to the other products.

Figure 5.5.4-4 assumes that all CSCIs are of equal importance. This may not be the case in reality. However, the manager can use the importance of each CSCI and the information in the figure to determine overall quality. If the preponderance of defects is limited to an unimportant CSCI, the total quality of the software from the customer's point of view may be good.

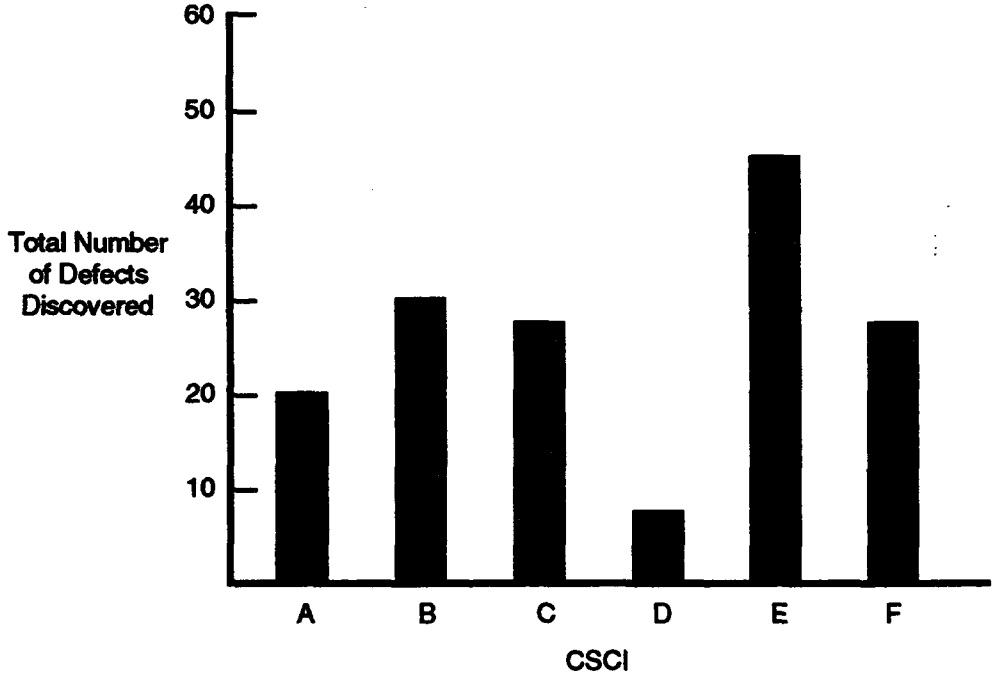


Figure 5.5.4-4. Distribution of the Number of Defects by CSCI

An alternate way of viewing the data is shown in Figure 5.5.4-5. Here the data are normalized to give defect density for each CSCI. The figure shows an upper limit on the defect density. The limit is based on historical data or is the project's or organization's quality goal. The example highlights that software quality assurance personnel or the manager needs to determine why CSCI D has a considerably lower defect density and CSCI E has a higher defect density than the other CSCIs. An unusually low defect density may indicate a high quality product or a poor quality peer review process.

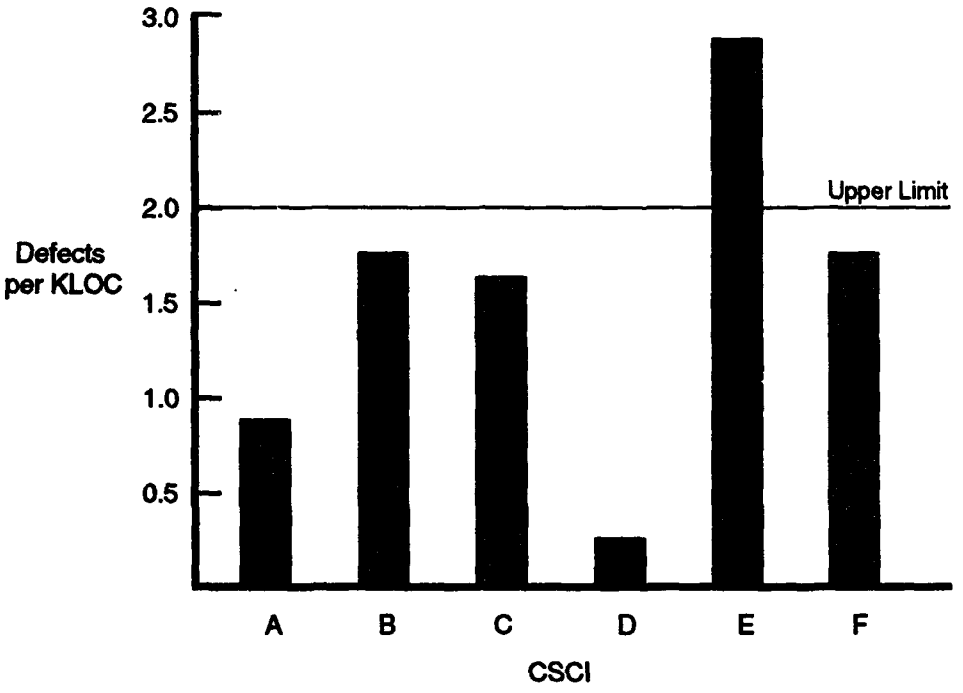


Figure 5.5.4-5. Defect Density for CSCIs

Figure 5.5.4-6 shows the defect density observed for each life-cycle activity. This figure is a plot of the total number of defects detected for a particular type of peer review (e.g., code review) during a particular build or release divided by the number of that type of review held during that life-cycle stage (e.g., the number of defects found during a peer review of products for requirements analysis divided by the number of requirements analysis peer reviews). The figure provides information on the effectiveness of the different type of reviews. If defect density is less in the early activities than in later activities, the SEPG may want to analyze the peer review process for the earlier activities since it is desirable to find defects early. Once data have been collected for multiple projects, they can be used to establish estimates of the number of defects expected. Detection of defects early in the life cycle may indicate a mature software process, but it may also indicate that the project has the personnel with the right skills and experience level working on the project at that time.

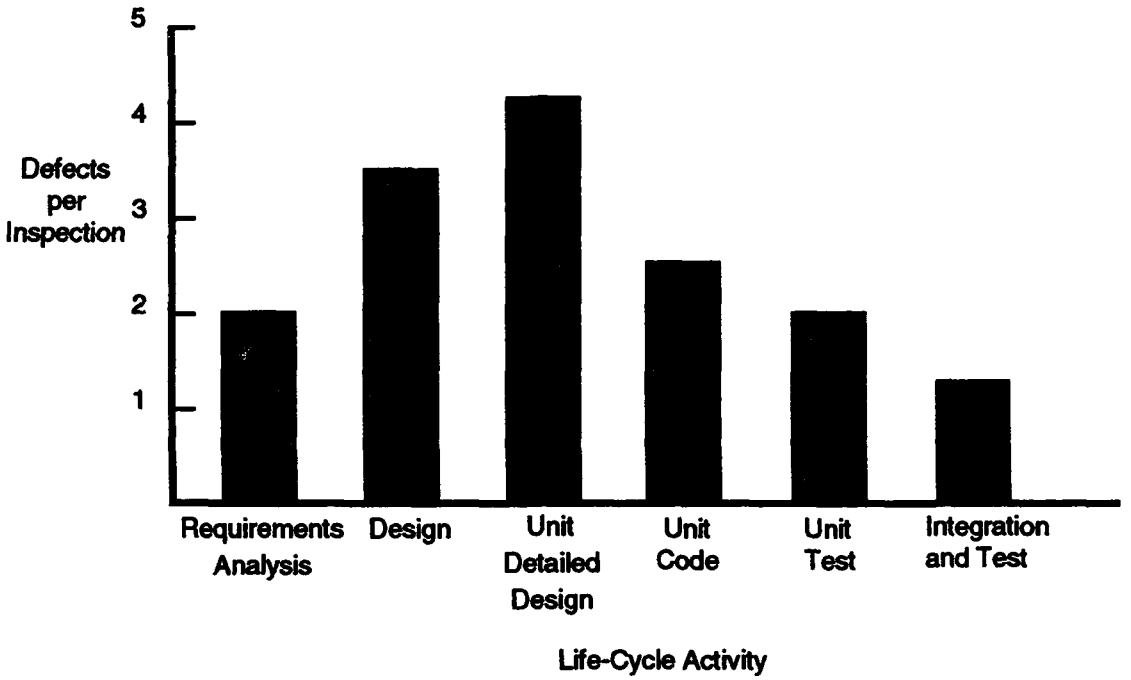


Figure 5.5.4-6. Defect Density for Each Life-Cycle Activity

As noted above, it is desirable to detect defects as early in the life cycle as possible. Figure 5.5.4-7 shows the number of requirements defects detected during each life-cycle activity. If the majority of such defects are found late in the life cycle, either the peer review processes occurring in the early stages or the requirements definition process needs to be improved. Similar figures can be drawn for defects detected during design and coding. The figure could also be plotted by using the percentage of requirements defects found in each life-cycle stage. The interpretation remains the same.

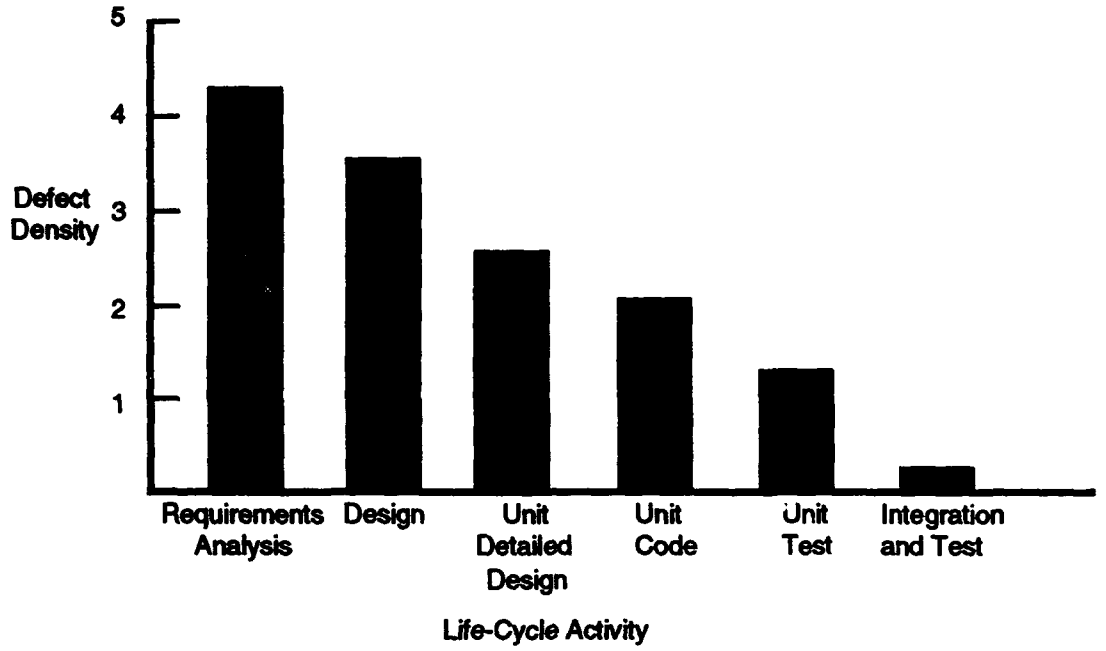


Figure 5.5.4-7. Density of Requirements Defects Found During Peer Reviews in Different Life-Cycle Activities

During a build or release, the project software manager can get an indication of the quality of the products under development by the number of items that required a second peer review. As part of the peer review process, acceptance criteria are established for each type of product. If the product does not satisfy these criteria, an additional review is required before the product can proceed to the next development stage. If the team is developing high quality products, the number of items requiring a second review is small. If a large percentage of the items have a second review, the reasons for this must be determined: the items are complex, the personnel doing the work do not have the necessary skills, the process needs improvement, and so forth. The ideal is to have no item re-reviewed, that is, the goal is a zero-defect product. Every defect discovered indicates rework which the project wants to minimize.

Figure 5.5.4-8 shows the re-review rate for designed units for a particular CSCI. The percentage is obtained by dividing the number of items that required a second review by the total number of initial reviews during the reporting period and multiplying by one hundred. The frequency of the figure is weekly, but depending on the project and its schedule, the manager can review such plots on a monthly basis. The figure also indicates the range within which the re-review rate can be expected to lie. This normal range is determined from historical data, when available. At the Managed Level, the upper and lower limits can be established using statistical techniques.

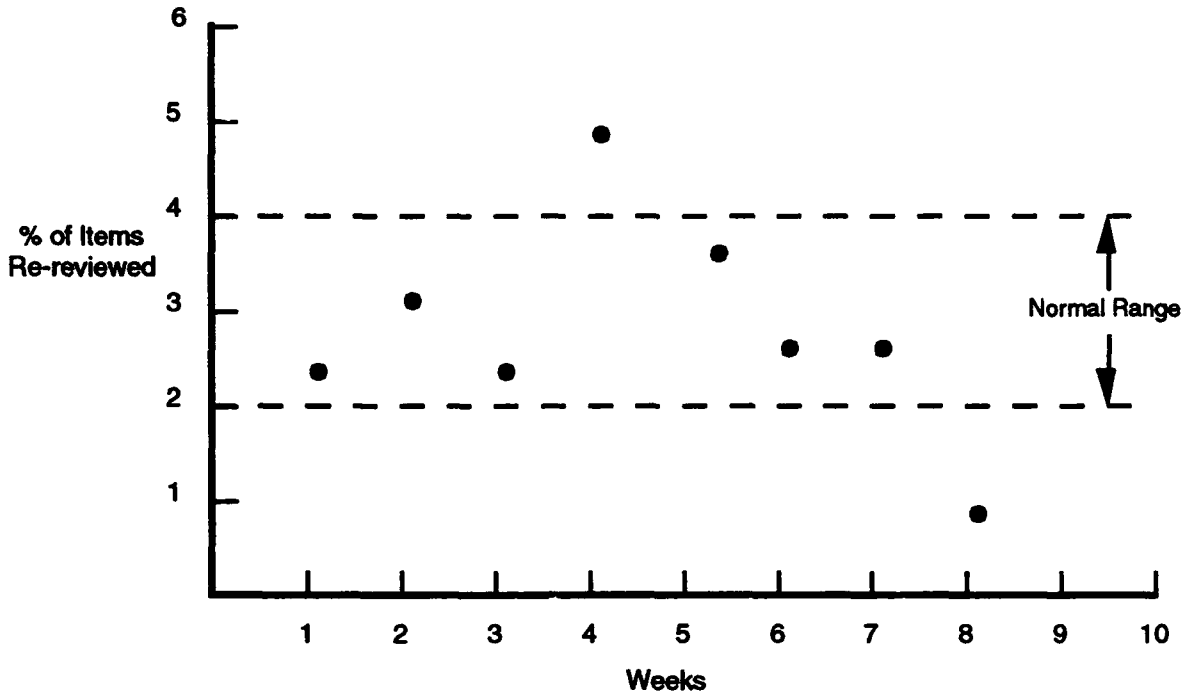


Figure 5.5.4-8. Percentage of Items Requiring a Second Review

Figure 5.5.4-9 shows the defect detection efficiency of the peer review process. The ratio of defects detected in peer reviews over the total number of defects detected is plotted over time for products from a project or all projects in an organization. The total number of defects detected is the number detected in both peer reviews and testing, therefore a datum on this figure cannot be plotted until testing is complete for the product. In an organization with a good peer review process, most defects will be detected before the test activity. In this example, the organization has made improvements in its peer review process. The results show that the organization is detecting significantly more errors before testing, when it is less expensive to remove them. However, the majority of defects continue to go undetected until testing.

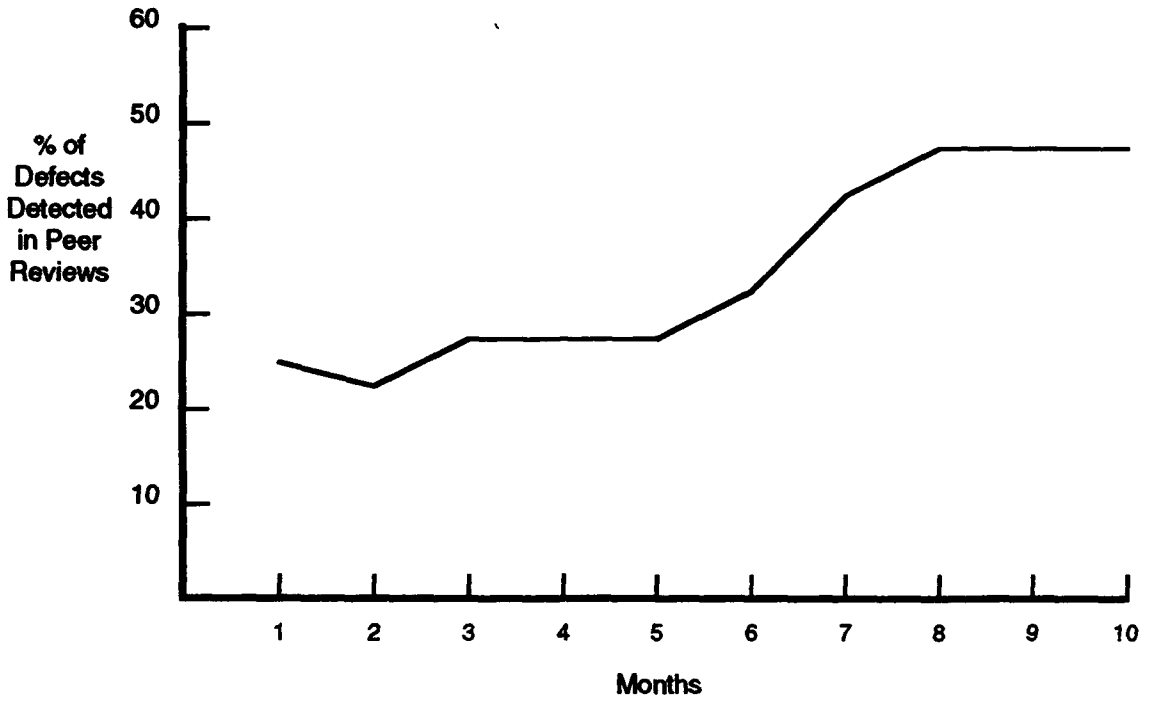


Figure 5.5.4-9. Peer Review Efficiency

Sources

[Decker 91] and [Florac 92] were the major sources of information for this section.

[AFSC 87], [Buckley 90], [Card 90], [Grady 87], [IEEE 1061], [Landis 90], [Pfleeger 89], and [Rozum 92] discuss concepts on tracking and analyzing trouble reports that are applicable to peer review data.

[IEEE 1044] was consulted for the types of inspection data to be collected.

[Pyzdek 89] discusses Pareto analysis.

5.6. Stability

At the Defined Level the stability indicators concentrate on the stability of the requirements, size, and process. Requirements stability is concerned with the number of changes to the requirements, the number of waivers from requirements, and the length of time the requests for changes to requirements are open. Size stability is concerned with code size and size estimates. Process stability is concerned with the number of changes to the software process and the number of waivers from the process.

5.6.1. Requirements Stability

The lack of requirements stability can lead to poor product quality, increased project cost, and/or lengthened project schedule. At the Defined Level, the project continues to use the requirements stability indicators of the Repeatable Level, but starts to investigate which categories of requirements are changing the most and to analyze the waiver requests from the contractual software requirements.

Objective of the Requirements Stability Indicators

To provide the project software manager and the software engineering process group (SEPG) with visibility into the nature and magnitude of requirements changes.

Indicators

From the Repeatable Level:

- Trends in the total number of
  - Requirements changes
  - TBDs

For the Defined Level:

- Trends in the total number of waivers from contractual software process requirements
- Length of time for analysis and action on change requests
- Amount of effort required to implement change requests

Key Process Area Goals Addressed

Integrated Software Management:

- The planning and managing of each software project is based on the organization's standard software process.
- Technical and management data from past and current projects are available and used to effectively and efficiently estimate, plan, track, and replan the software projects.

Software Product Engineering:

- The software engineering activities are well-defined, integrated, and used consistently to produce software systems.
- State-of-the-practice software engineering tools and methods are used, as appropriate, to build and maintain the software system.

Life-Cycle Stages: All, most important during requirements and design

Users

From the Repeatable Level:

- Software engineering manager for control and monitoring of requirements
- Software engineering process group for sources of and reasons for instability

For the Defined Level:

- Project software manager

Users' Questions

From the Repeatable Level:

- Is the number of changes to the requirements manageable?
- Are the requirements scheduled for implementation in a particular release actually addressed as planned?
- Is the number of requirements changes decreasing with time?
- Is the number of TBDs preventing satisfactory completion of the product?
- Is the number of TBDs decreasing with time, that is, are the TBDs being resolved in a timely manner?

For the Defined Level:

- What categories of requirements are responsible for the majority of requirements changes?
- How does the number of requirements changes on this project compare to past, similar projects?
- How responsive are the staff addressing requirements change requests?
- How does the number of waivers of requirements on this project compare to past, similar projects?
- What is the impact of requirements changes on effort and cost?

Input

From the Repeatable Level:

- Total number of requirements

- Number of requirements changes:
  - Proposed
  - Open
  - Approved
  - Incorporated into baseline
- For each requirements change:
  - The computer software configuration item(s) (CSCI) affected
  - Major source of request (customer, software engineering, etc.)
  - Requirement type (functional, performance, etc.)
- Number of TBDs in requirements specifications
- Number of requirements scheduled for each software build or release

For the Defined Level:

- Date of change request
- Date change request approved
- Effort and cost to analyze the proposed changes and to achieve agreement
- Size and cost to implement and test, including initial estimate and actuals (for major changes only)
- Number of waivers for deviations from contractual software process requirements:
  - Number requested
  - Number approved
  - Number rejected

Interpretation

Figure 5.6.1-1 shows the number of changes to requirements and the range of requirements changes observed on past, similar projects. Ideally, the number of requirements changes decreases as the project advances through the life cycle. This is evidenced by the smaller range in changes at critical design review (CDR) than at the system design review (SDR). The project software manager can track how the project is doing compared to other past, similar projects. In this case, the project is very similar to others. If the project lies outside the range, the manager needs to determine the cause. For an excessive number of changes, the requirements specification may be of poor quality, and the manager and the customer need to address this issue. If the actual number of requirements changes tracks closely to the planned profile, the project software manager can use the figure to predict how many more requirements changes the project can expect. This information can be used with the progress and effort indicators to check the current estimates for progress and effort. When analyzing Figure 5.6.1-1, the project software manager needs to be cognizant of the size of each change.

One change can have a large impact on the cost, effort, and schedule. Conversely, several small requirements changes can have little impact on cost, effort, and schedule.

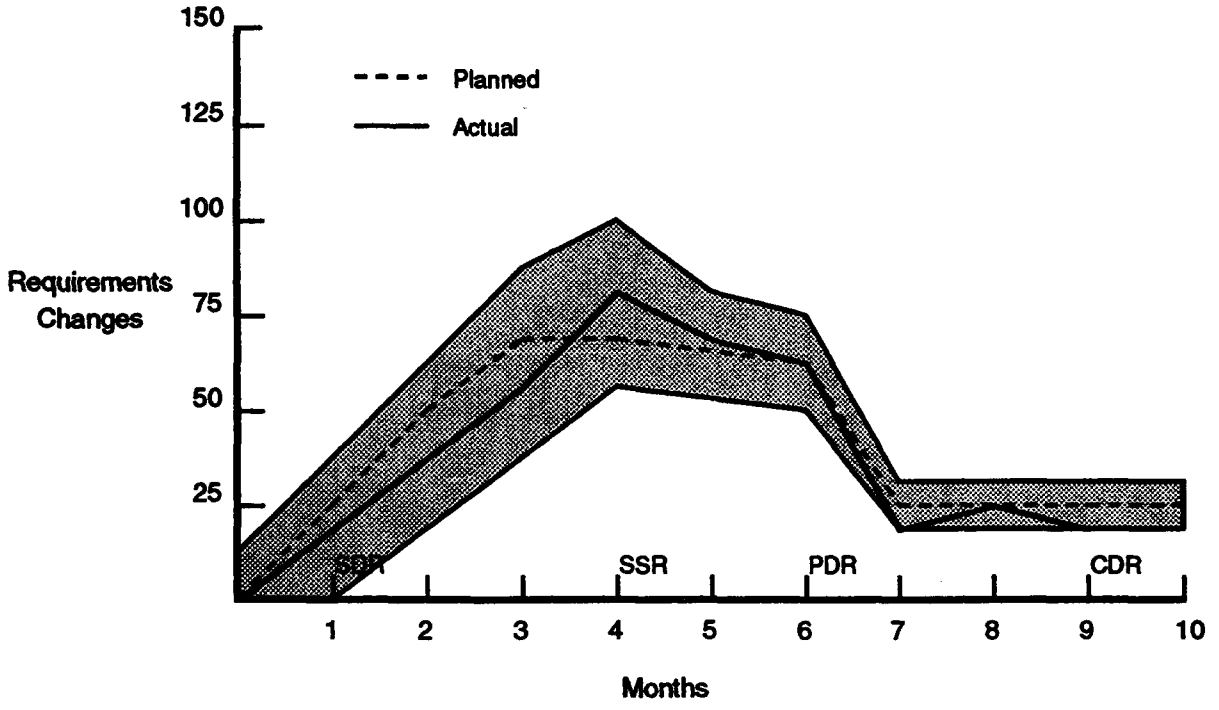


Figure 5.6.1-1. Total Number of Requirements Changes

Figure 5.6.1-2 shows the total number of requirements changes approved to date and breaks this total into component requirements types. The project manager, the SEPG, or software engineering personnel can determine whether any one type is dominating or chiefly responsible for the requirements changes. If the figure shows that the majority of the total changes is due to one category, then the cause needs to be determined. Did the process that produced the requirements break down? Does the process need to be changed? Would a change to the process help? The types in the figure are chosen as representative of the types of requirements and are not complete.

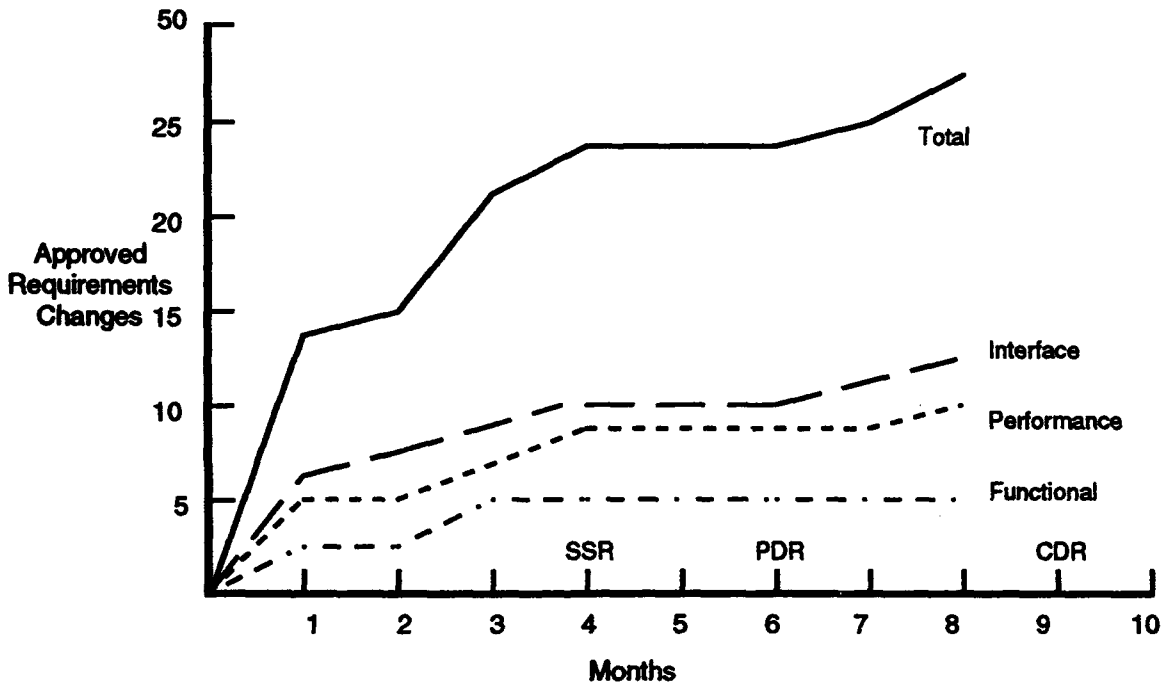


Figure 5.6.1-2. Total Number of Requirements Changes and Number of Changes by Requirement Type

Figure 5.6.1-3 shows the total number of waiver requests on the project to-date as well as the number requested per reporting period. The manager needs to compare the numbers on the project with the historical data to determine whether the numbers are within bounds. The manager may wish to plot only the approved waivers or the total number requested and the approved number. If the number requested or approved is excessive, the manager, the SEPG, or software engineering personnel need to determine the cause for the high numbers. If the number requested or approved is lower than the normal range, the same groups monitor the products to ensure that the project personnel understand the requirements. Well-written requirements can result in a lower number of waiver requests.

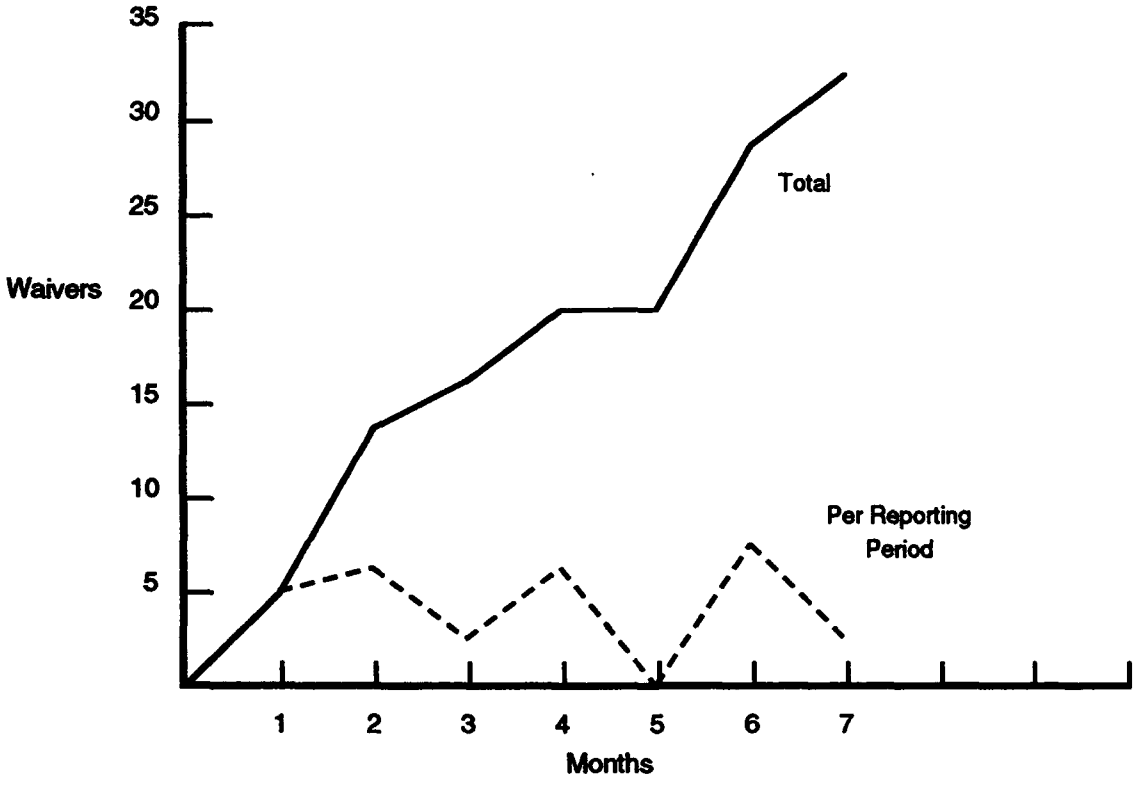


Figure 5.6.1-3. Cumulative Number of Waivers and Number per Reporting Period

Figure 5.6.1-4 shows the number of change requests that have been analyzed and acted upon per time interval. It shows how much time is required for a change request to be analyzed and acted upon (either approved or rejected). The main indicator in which the manager or software engineers are interested is the amount of time required for an "average" or typical change request to be resolved. Minor changes require less time than complex changes. From a process point of view, software engineers want to minimize the amount of time required to analyze the request, and the manager wants to minimize the control process without jeopardizing the quality of the analysis and the control board deliberations.

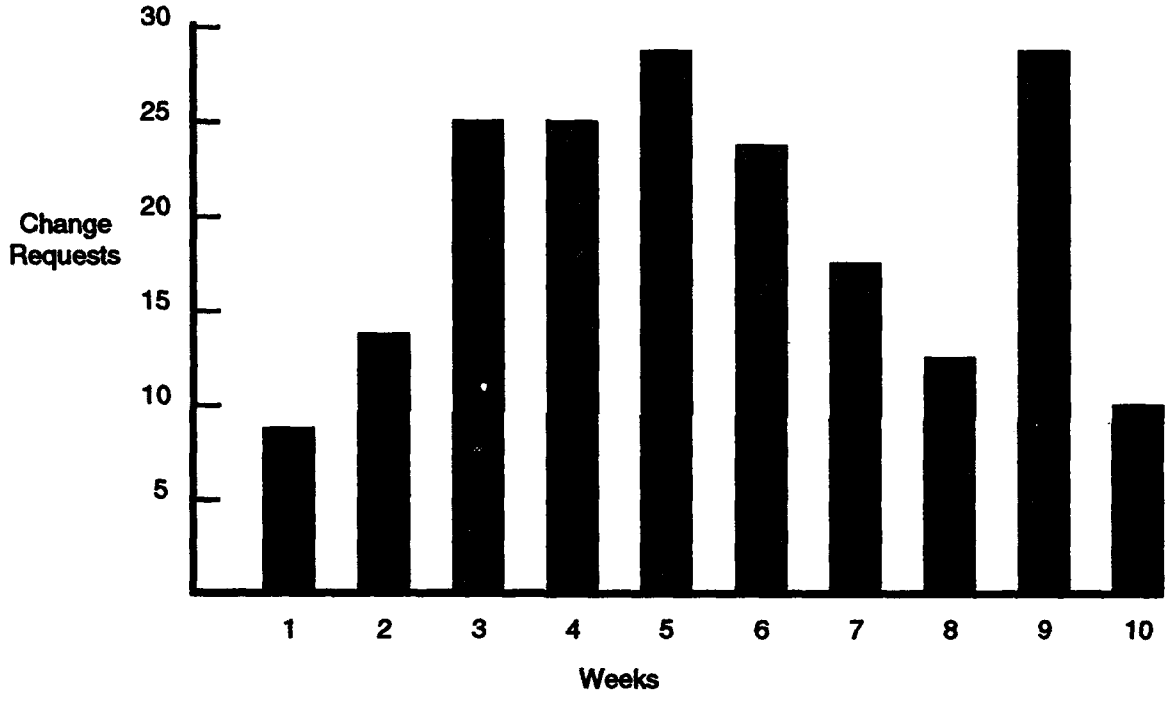


Figure 5.6.1-4. Length of Time for Change Request Analysis and Action

A figure similar to Figure 5.6.1-4 can be prepared that shows the amount of effort or magnitude of the change for change requests, that is, the figure would show the number of change requests by either the size of the change (measured in the number of units or lines of code, for example) or the amount of effort required to make the approved change. The analysis of this figure would be similar to that of Figure 5.6.1-4.

Sources

[AFSC 86] has a similar indicator that uses software size.

[Decker 91] discusses a software volatility/software requirements measure.

[Grady 87] uses a requirements stability as an input to Hewlett-Packard's difficulty metric.

[Landis 90] has two indicators that are related to this but use software size in their trend chart.

[Pfleeger 89] discusses a requirements volatility metric.

[Schultz 88] discusses a software volatility metric.

5.6.2. Size Stability

The indicators for size stability are the same as those at the Repeatable Level. At the Defined Level, managers can use historical data from other similar projects that allow them not only to track actual size against the planned size, but also as a frame of reference for making the original size estimates. This frame of reference is in the form of a range centered on the planned size. Any variation from the plan that is still within this range is considered normal. Variations outside this normal range require analysis by the manager.

Objective of the Size Stability Indicators

To provide the project software manager and the project manager with an indication of the completeness and stability of the requirements and of the capability of the implementation staff to produce the software product within the current budget and schedule.

Indicators

Same as the Repeatable Level:

- Trends in the code size
- The variation of actual software size from size estimates
- Variation of actual software size from estimated size by build or release

Key Process Area Addressed

Integrated Software Management:

- Technical and management data from past and current projects are available and used to effectively and efficiently estimate, plan, track, and replan the software projects.

Life-Cycle Stages: All

Users

Software engineering and project software managers for controlling and monitoring of project

Users' Questions

From the Repeatable Level:

- How much have the size estimates changed over time during development?
- How much do the actual values deviate from their estimated values?
- How much does the trend of the actual values affect the development schedule?
- Is the estimated productivity sufficient to allow the completion of added code on schedule or are more staff required?

Input

From the Repeatable Level:

- Software size estimates:
  - Total size
  - By computer software configuration item (CSCI)
  - New code
  - Off-the-shelf code
  - Reused code
- Amount of software scheduled for completion by build or release

Interpretation

Figure 5.6.2-1 shows the change in software size over time for the project and also in comparison with similar projects. The interpretation of the figure remains the same as that of the Repeatable Level except for the comparison with the historical data. This figure is similar to Figure 4.6.2-1, but the range of software sizes determined from historical data for similar projects has been added, and the size of the new and reused software has been omitted. This comparison allows the manager to determine how the project is tracking against the "norm." If deviations from the norm occur, the manager needs to determine the reason.

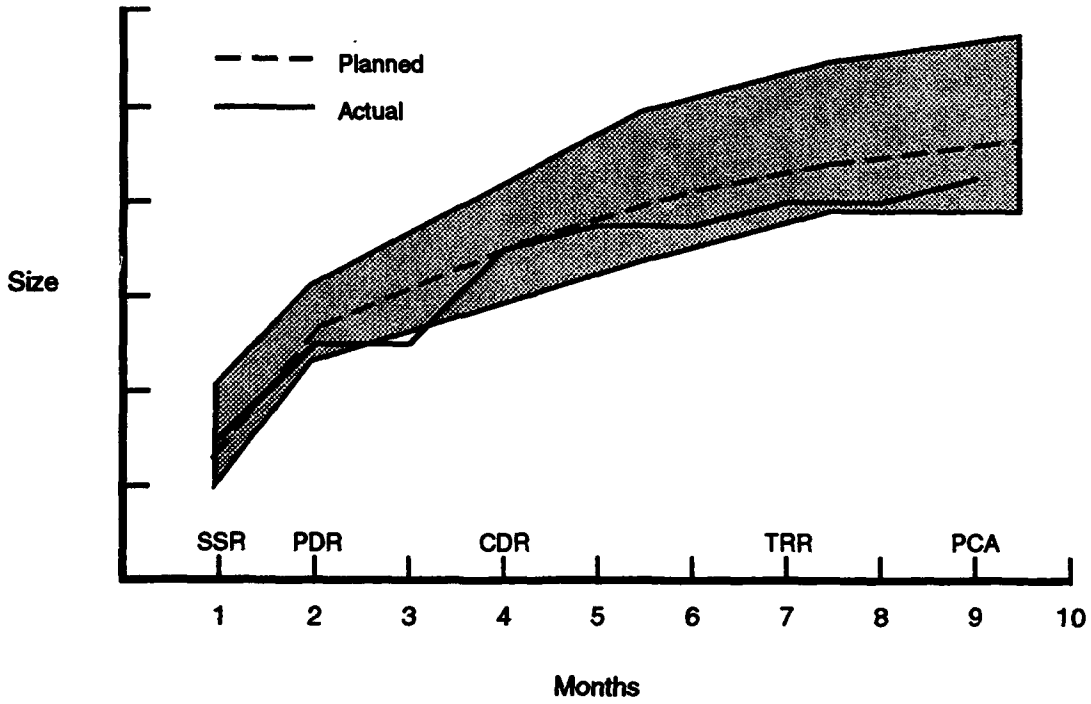


Figure 5.6.2-1. Software Size as a Function of Time Compared to Similar Projects

Figure 5.6.2-2 shows an alternate way to analyze the size growth data. The figure is based on the work of Landis et al and shows the Software Engineering Laboratory (SEL) Size Estimate Model [Landis 90]. Growth is expressed as a percentage and is determined at key points in the life cycle. Their data show that as the details of the unknown portions of the requirements (the to-be-determineds) become known, the size growth grows more rapidly. Hence, the range of accepted growth narrows as the system becomes better defined.

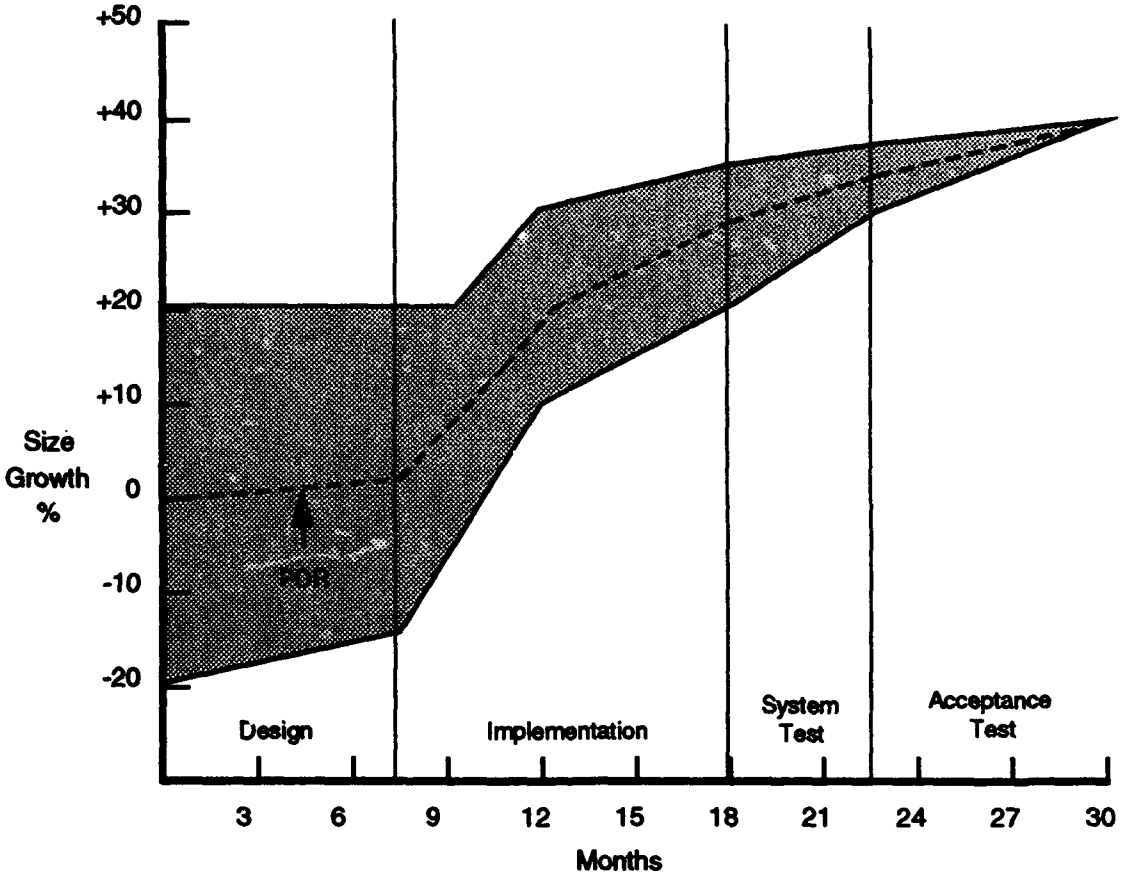


Figure 5.6.2-2. Software Size Growth

If the actual growth is greater than planned, the manager can look for an incomplete design (especially during implementation) or numerous changes in the requirements. The manager can use the requirements stability indicator for the latter. If there is little or no growth in the size estimate after the preliminary design reviews, the project may be fortunate to have an experienced team that is familiar with the application and/or has a set of stable, well-defined requirements. The manager can use the requirements stability indicator or the effort indicator to verify these assumptions.

Figures 5.6.2-1 and 5.6.2-2 show the data for the entire project. They can also be drawn for each CSCI.

Sources

From the Repeatable Level:

[AFSC 86], [Decker 91], [Landis 90], [Pfleeger 89], and [Schultz 88] all discuss tracking software size.

5.6.3. Process Stability

A characteristic of the Defined Level is a well-defined process whereby project personnel perform their work. The process stability indicators provide information on the stability of the process by monitoring the number of requests to change the process and the number of waivers to the process. A large number of requests may indicate that the defined processes are not as efficient as believed or may not be appropriate to a particular project. A large number of changes or waivers granted may indicate that the processes are actually undergoing ad hoc revision and are not as stable as believed.

Objective of the Process Stability Indicators

To provide an indication of the stability of the defined process and to provide the software engineering process group (SEPG) with an indication of the quality of the defined process.

Indicators

Trends in the number of the following:

- Software process change requests
- Waivers to the software process requirements

Key Process Area Goals Addressed

Organization Process Definition:

- A standard software process for the organization is defined and maintained as a basis for stabilizing, analyzing, and improving the performance of the software projects.

Integrated Software Management:

- The planning and managing of each software project is based on the organization's standard software process.

Software Product Engineering:

- Software engineering issues for the product and the process are properly addressed in the system requirements and system design.
- The software engineering activities are well-defined, integrated, and used consistently to produce software systems.
- State-of-the-practice software engineering tools and method are used, as appropriate, to build and maintain the software system.

Life-Cycle Stages: All

Users

- Project software manager
- SEPG
- Senior management

Users' Questions

- How stable is the process?
- Are process problems due to a poor process definition or poor implementation?

Input

- Process plan
- Process change requests:
  - Number open
  - Number approved
  - Number rejected
  - Number incorporated
- Waiver requests:
  - Number approved
  - Number rejected
- For each process change or waiver request:
  - Source of request (e.g., group, department)
  - Impact of request (e.g., minor or major)

Interpretation

Figure 5.6.3-1 shows the total number of change requests and the number of approved requests per reporting period. A high number of change requests indicates that the personnel do not sufficiently understand the process or that the standard process may be inappropriate and should be revised. A high number of approved change requests indicates that the process may be unstable. An objective, thorough analysis is necessary to determine the reasons for the high approval rate. The figure can also be plotted with cumulative change requests.

Not shown on the figure are the numbers of requests that have been approved and not incorporated. Process changes must be carefully managed just like requirements changes. A change to a process can be approved, but the appropriate planning for introducing the change must occur. Minor changes can be saved and introduced organization-wide quarterly, semiannually, or whenever appropriate. Major changes, of course, can be made as soon as possible after approval if the organization is ready (e.g., there is sufficient funding, training, staffing, etc.).

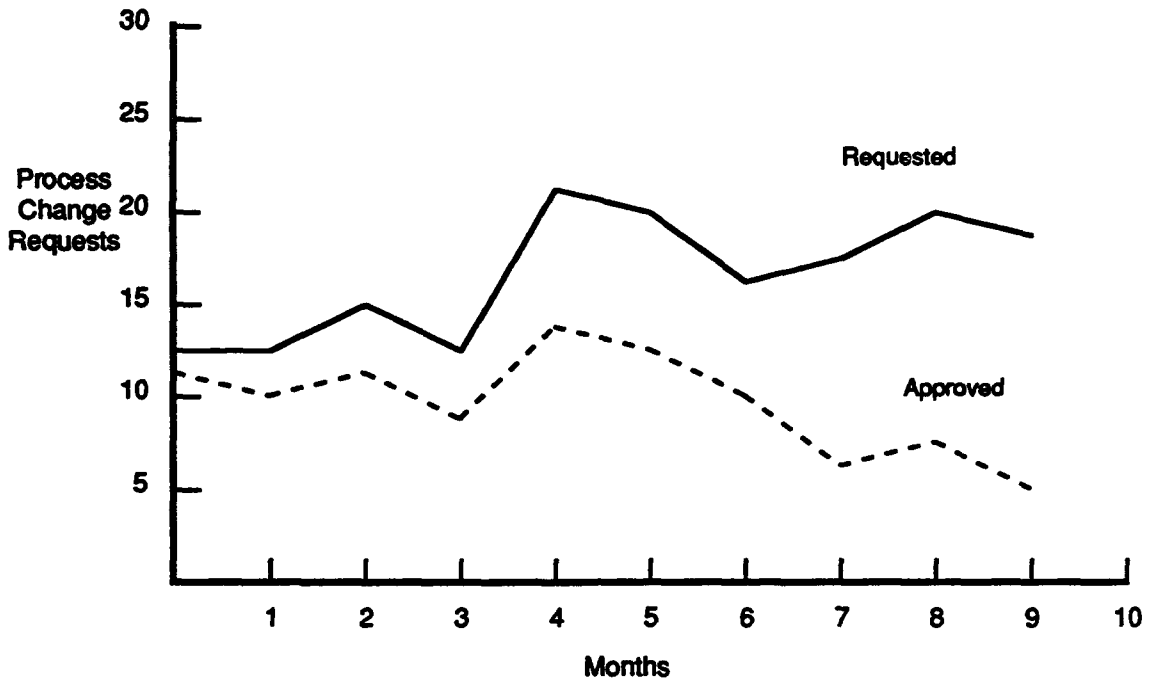


Figure 5.6.3-1. Process Change Requests

Figure 5.6.3-2 shows the total number of requests for waivers and the number of approved requests per reporting period. A high number of requests for waiver indicates that the process may be inappropriate or that the process may be appropriate but is not designed for flexibility. A high number of approved waivers indicates that the process may be unstable and may in fact be changing in an uncontrolled manner. The figure can also be plotted with cumulative change requests.

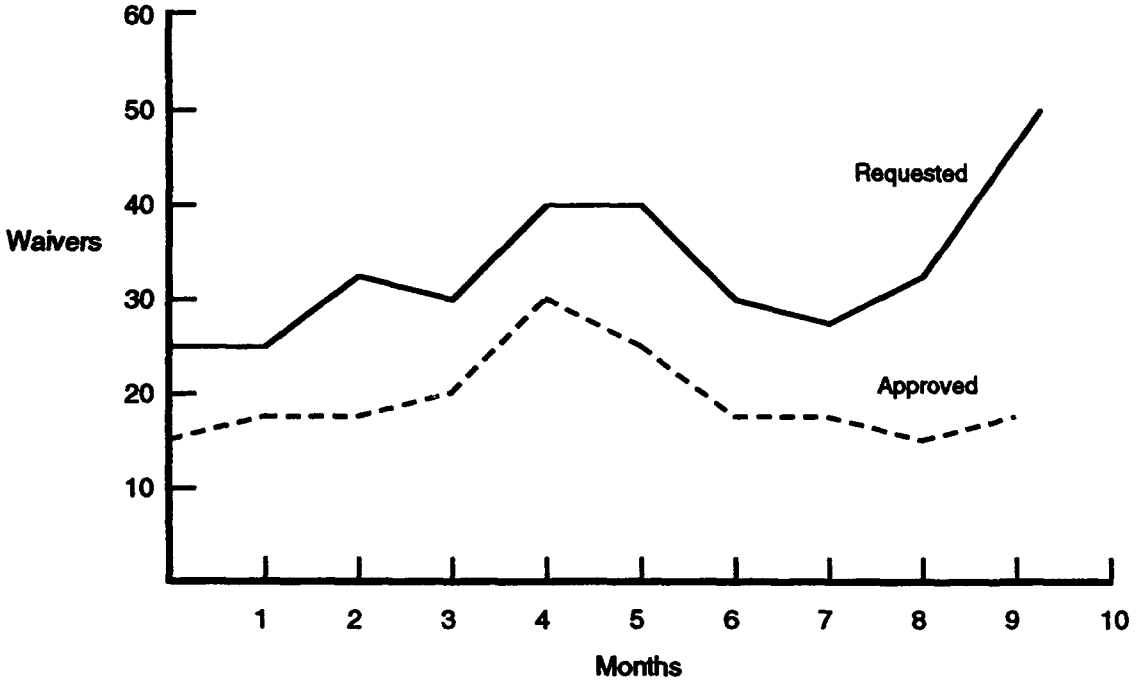


Figure 5.6.3-2. Waivers from Process Standards

Sources

[Humphrey 89] discusses the role of the SEPG in process development and monitoring.

5.7. Computer Resource Utilization

The indicators for computer resource utilization are the same as those at the Repeatable Level (see Section 4.7.). At the Defined Level these indicators address the Integrated Software Management key process area.

5.8. Training

A trained staff is a valuable asset for the project software manager. The manager ensures that the staff has the skills to perform its assigned tasks. The training indicator provides the manager with information on the training program.

Objective of the Training Indicators

To provide managers visibility into their training process, to ensure effective utilization of training, and to provide project software managers with an indication of their staff's mixture of skills.

Indicators

- Deviations in the number of the following:
  - Classes taught from the number of classes planned
  - Staff taught from the number planned
- Quality of the training program
- Number of waivers from training:
  - Requested
  - Approved

Key Process Area Goals Addressed

Training Program:

- The staff and managers have the skills and knowledge to perform their jobs.
- The staff and managers effectively use, or are prepared to use, the capabilities and features of the existing and planned worked environment.
- The staff and managers are provided with opportunities to improve their professional skills.

Life-Cycle Stages: All

Users

- Project software manager for the details on the project
- Senior management for consolidated cost figures of the organizational training program
- Software engineering process group for appropriateness and effectiveness of training program

Users' Questions

- Who is attending training?
- Are the appropriate people being trained?

- How many of the staff are being trained?
- How many of the staff have completed their training?
- Does the staff have the appropriate mixture of skills for the project?
- How is the project performing with respect to its training plan?
- What is the quality/effectiveness of the training?
- How many waivers from training have been requested?
- How many waivers from training have been approved?
- What is the cost of training (total and per capita)?

Input

- Training plans
- Number of classes actually taught
- Attendance records
- Types of people trained (e.g., technical, support personnel, managers)
- Training survey results
- Waiver requests:
  - Number of requests (by course and by project)
  - Number approved (by course and by project)
- For each waiver request:
  - Reason for request
  - Source of request (e.g., skill level of staff, project, department)

Interpretation

In the discussion that follows the terms "course" and "class" are used. A course is an offering in a particular subject, for example, a course in requirements analysis. A class is the number of offerings of a particular course. For example, there may be one course in requirements analysis, but there are six classes in one month for that course. There are also different levels of courses. They range from an overview of a particular subject to an in-depth discussion of that subject. The organization needs to determine what the goals of its training program are and develop the appropriate courses, whether they are overview, refresher, or in-depth courses. The organization also needs to determine the number of classes required to meet the demand for training.

Figure 5.8-1 shows the planned number of classes offered versus the actual number offered in each reporting period. There is a chart for every course. Deviation from the plan may indicate a lack of accurate planning or commitment to the training program. Since training affects skill mix, a project's mixture of skills may not be what it should be. Deviation may also reflect lack of available instructors, a decline in funds for training, or an insufficient number of students requiring the course.

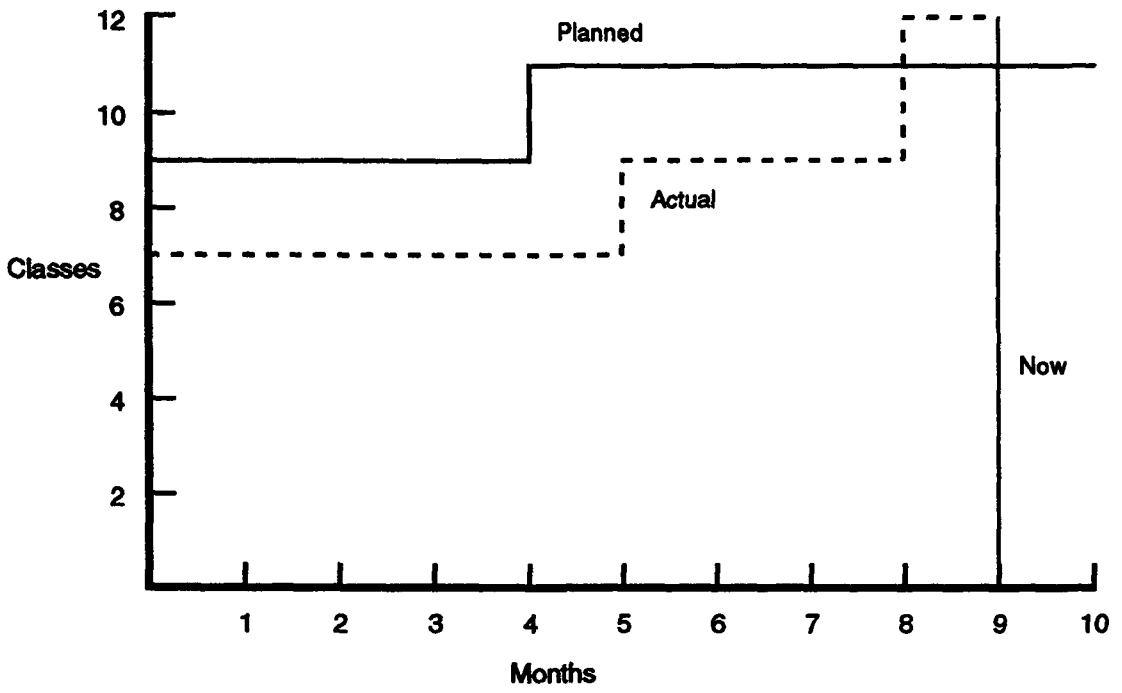


Figure 5.8-1. Number of Classes Offered Each Month

Figure 5.8-1 is drawn for the total number of classes. The number of courses should also be monitored to ensure that the appropriate subject matter is being offered. It is important to maintain the appropriate skill level needed for a project. For example, prior to each life-cycle stage, the project software manager wants to make sure that the staff has the appropriate skills for the next stage. Or if an organization determines that it will introduce a new technology, the requisite training must occur before the staff is expected to use that new technology.

Figure 5.8-2 shows the planned attendance versus actual attendance. There is a chart for every course. Deviation for the plan may indicate lack of time, money, or interest of participants. Since training affects the skill of the staff, the project's mixture of skills may not be what it should be.

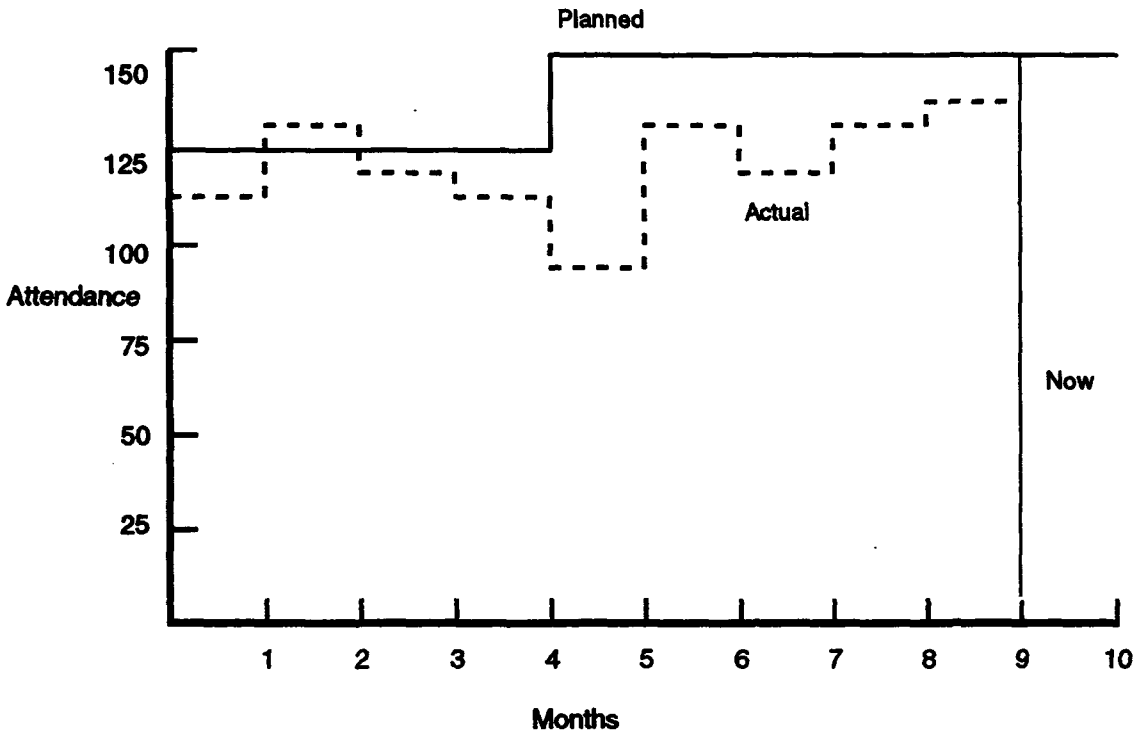


Figure 5.8-2. Total Attendance

Figure 5.8-3 shows an example plot that depicts the quality of a course. The x-axis is labeled with example quality attributes. The training plan defines which attributes are collected and tracked. The y-axis runs from 1 to 5: the higher the number, the better the attribute. There is a chart for every class of every course, if appropriate. Low scores indicate that personnel are not receiving the training needed for the project. A low instructor rating may indicate an instructor who lacks in-depth knowledge of the material. Such courses may be flagged for an improvement effort .

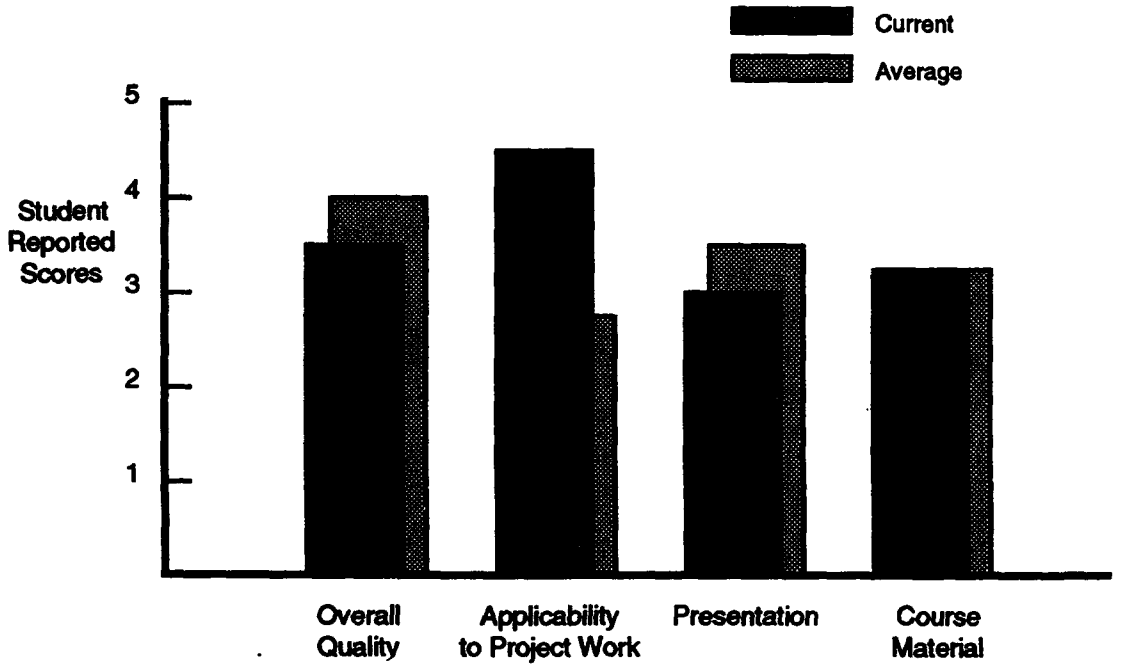


Figure 5.8-3. Course Quality

Figure 5.8-4 shows the number of waivers requested and the number approved. There should be a chart for every project, course, or labor category where appropriate. A high number of requests may be indicative of a low quality, ineffective, or inappropriate course. If so, the training plan should be reevaluated. A high number of approved requests also indicates that the training plan has become unstable or unenforced. This may lead to a mismatch in a project's mixture of skills.

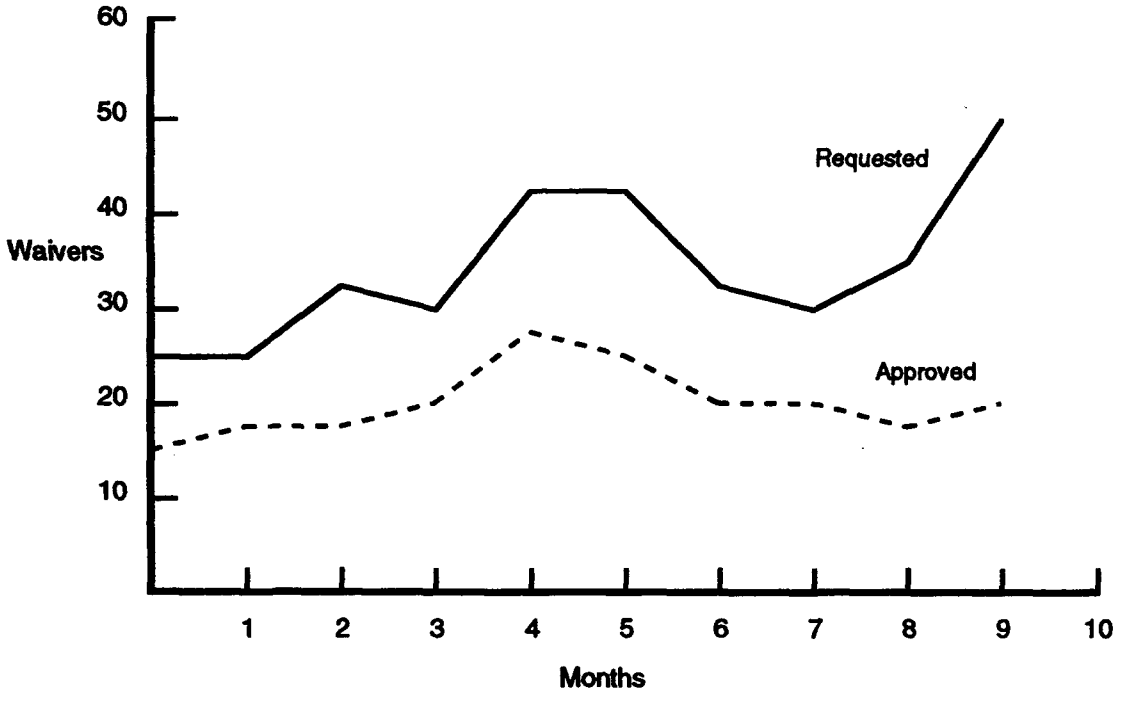


Figure 5.8-4. Waivers from Training Courses

Sources

[London 89] discusses high-quality, cost-effective employee training in organizations.

6. The Managed Level—Maturity Level 4

This chapter summarizes the characteristics of an organization with a managed process, provides a brief introduction to statistical process control, and discusses the indicators that are appropriate for the Managed Level.

6.1. Characteristics of a Managed-Level Organization

An organization with a managed process is characterized as one that sets quantitative quality goals for software products, measures productivity and quality for important software process activities across all projects in the organization, and uses a process database to collect and analyze data. A Managed-Level organization uses a set of well-defined and consistently-defined measures to establish acceptable quantitative boundaries for product and process performance. This allows meaningful performance variation to be distinguished from random variation.

Indicators appropriate for a Managed-Level organization are progress, effort, cost, quality, stability, computer resource utilization, and training. As noted in Chapter 1, there are few examples of proven indicators for this maturity level. Some discussion of indicators in this chapter is based on expected practice. As organizations mature, experience with indicators at this maturity level will increase, and the information in this chapter could change.

6.2. Statistical Process Control—An Overview

At the Defined Level, the concept of ranges of the planned items was introduced. At that level, historical data are used to select a range around the plan based on some percentage of the planned value. At the Managed Level, however, the organization can use statistical methods to establish ranges and set the limits of variation around the plan. The organization makes extensive use of statistical process control (SPC) techniques.

Pyzdek defines SPC as the use of statistical methods to identify the existence of special causes of variation in a process [Pyzdek 89]. SPC has a basic rule: variation from common cause systems should be left to chance, but special causes of variation should be identified and eliminated. In other words, an organization uses SPC to identify when a process is out of control by looking at control charts to determine when an observed variation from the plan is due to special cause and not normal, random variation.

Figure 5.5.4-8 shows an example of a control chart for the percentage of items re-reviewed during each reporting period. In the figure, the manager visually determined the control limits. Most of the values were around three percent, so the manager established ±1 percent control limits around the three percent value. There is no firm basis for this assumption other than "it looks reasonable." In SPC, the manager can

determine an average value and the standard deviation. According to the normal distribution, 99 percent of all values lie within ±3 standard deviations. SPC uses this fact to establish the upper and lower control limits as three standard deviations above and below the average values, respectively. Log-normal distribution can be used to calculate the lower control limit so that it does not go below zero. Likewise, for figures that have a natural upper bound (e.g., one hundred percent), beta distribution can be used to prevent the upper control limit from going above the boundary.

Figure 6.2-1 summarizes the concepts attached to a control chart. Normal fluctuations are those that lie within the control limits, that is, they are natural variations. Any value lying above the upper control limit or below the lower control limit may indicate a process problem. The cause of the variation must be determined.

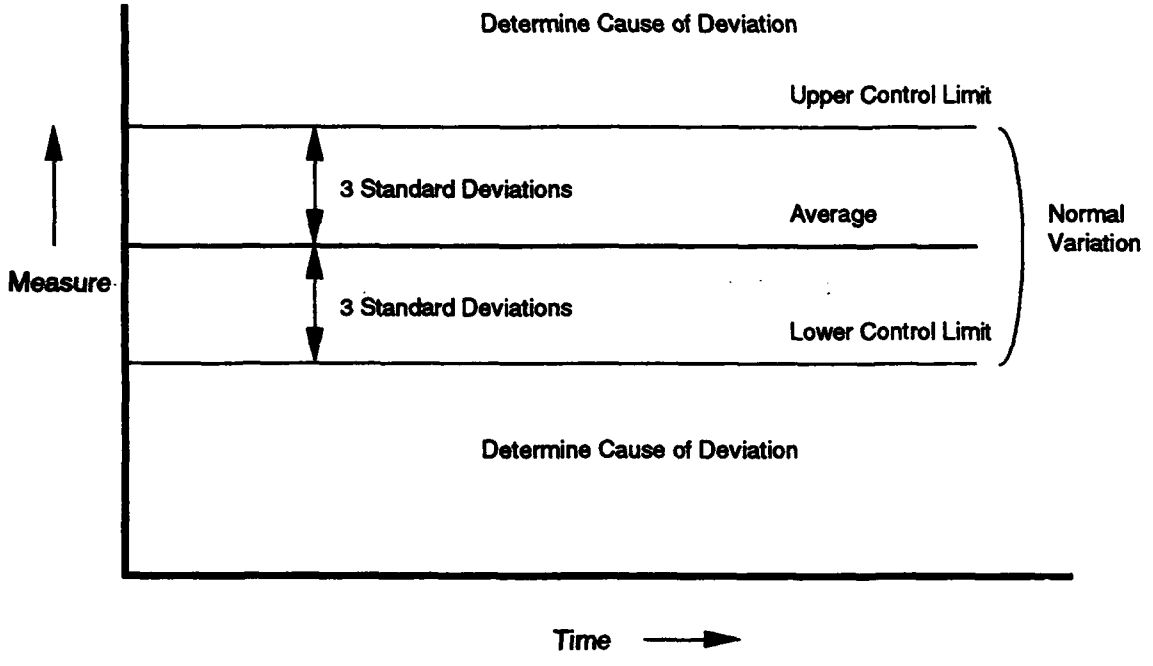


Figure 6.2-1. Control Chart

According to Pyzdek [Pyzdek 84], the following criteria indicate when a process is out of control:

- Any value exceeds a three standard deviation control line
- Four of five consecutive values exceed the ±1 standard deviation line
- Two of three consecutive values exceed a ±2 standard deviation line
- S even or more consecutive values lie on same side of the average

Figures 6.2-2 and 6.2-3 show the defect rate (defects per thousand source lines of code) for each release of two projects, one in SPC (Figure 6.2-2) and one out of SPC (Figure

6.2-3). Both projects have the same control limits (the lower limit is zero). The project in control has a defect rate that is within the control limit for each release. The second project was in control until release seven when there was a dramatic increase in the defect rate. The project is obviously out of control. Such a situation can occur when a release becomes so large that an unreasonable productivity rate has to be achieved to complete the release on schedule, and product quality is sacrificed to maintain the schedule.

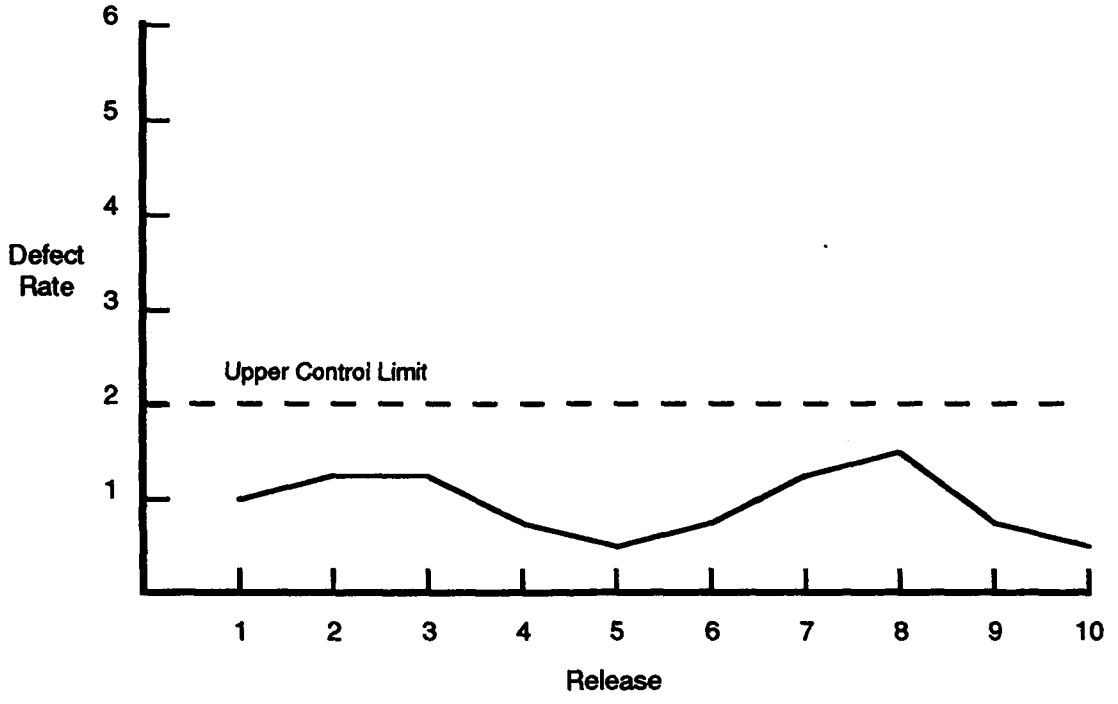


Figure 6.2-2. Project in Statistical Control

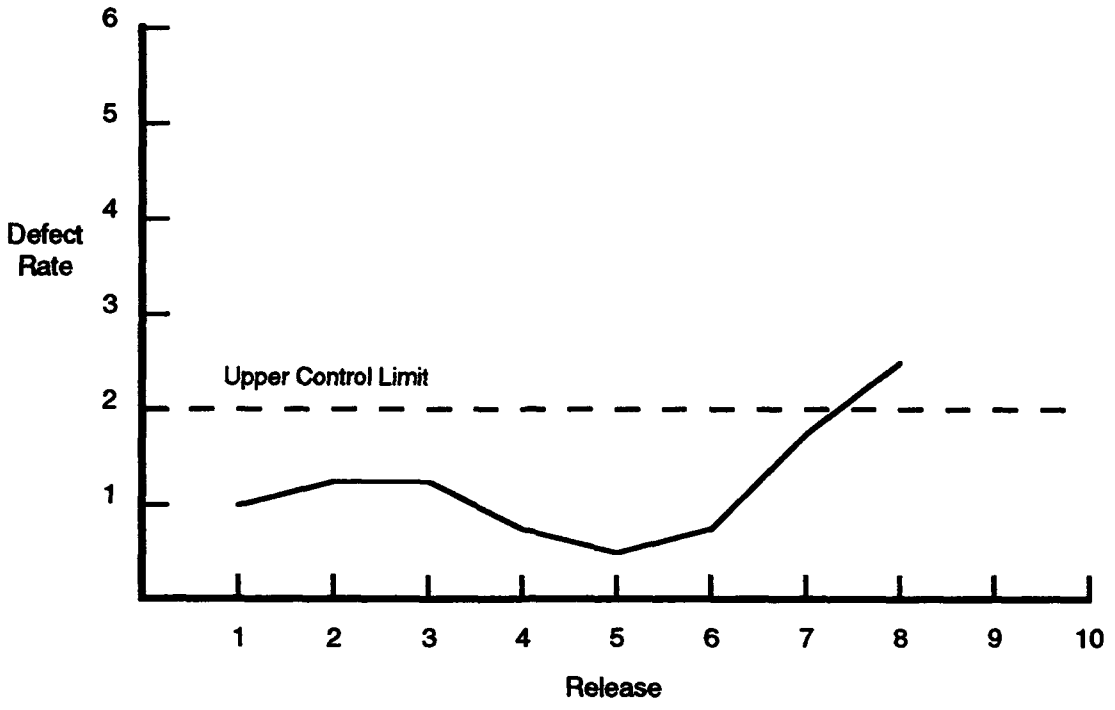


Figure 6.2-3. Project out of Statistical Control

6.3. Progress

The progress indicators for the Managed Level are the same as those at the Defined Level (Gantt and PERT charts and actual-versus-planned-completions charts). (See Section 5.2.) The difference between these two levels is in the application of SPC to the actual-versus-planned-completion charts. At the Defined Level, the range was set at some percentage of the planned completion. At the Managed Level, the project software manager uses historical data and statistical calculations to establish the control limits.

6.4. Effort

The effort indicators for the Managed Level are the same as those at the Repeatable and Defined Levels (see Section 4.3 and Section 5.3). In addition, the organization may wish to consider monitoring the following:

- Non-cumulative staff-hours by life-cycle activity over time
- Separately needed or special development skills
- Experience level with domain/application
- Experience level with development architecture
- Experience level with tools/methods over time
- Productivity of staff against the years of experience

6.5. Cost

The cost indicators for the Managed Level are the same as those at the Defined Level, that is, cost/schedule status and variance reports. The difference between these two levels is in the application of statistical process control to the cost/schedule status report. At the Defined Level, the control limits were set at some percentage around the line representing budgeted cost for work scheduled. At the Managed Level, the project software manager uses historical data and statistical calculations to establish the control limits. For the monitoring of the cost performance, the manager determines the cause when the budgeted cost for work performed and the actual cost for work performed fall outside the control limits.

Also at the Managed Level, the cost of quality is measured in addition to the cost of other project elements.

Objective of the Cost Indicator

To track actual costs against the project plan and to predict future project costs.

Indicators

From the Defined Level:

- Performance of the actual cost of work performed against budgeted cost for work scheduled
- Performance of the budgeted cost of the work performed against the budgeted cost for work scheduled
- Trends in the cost variance
- Trends in the schedule variance
- Trends in the cost and schedule performance indices

Key Process Area Goals Addressed

Process Measurement and Analysis:

- The organization's standard software process is stable and under statistical process control.
- The relationship between product quality, productivity, and product development cycle time is understood in quantitative terms.
- Special causes of process variation (i.e., variations attributable to specific applications of the process and not inherent in the process) are identified and controlled.

Quality Management:

- Measurable goals for process quality are established for all groups involved in the software process.

- The software plans, design, and process are adjusted to bring forecasted process and product quality in line with the goals.
- Process measurements are used to manage the software project.

Life-Cycle Stages: All

Users

All levels of management. The managers cited here are not restricted to software development managers but include managers of support groups, for example, software quality assurance, software configuration management, training, and so forth.

Users' Questions

From the Defined Level:

- Are the actual costs and performance of the software project tracking against the plan?
- Are the actual costs within the range established for the software project?
- Are the corrective actions taken bringing the actual cost of work performed and the budgeted cost for work performed back within the low-risk range?

Input

From the Repeatable Level:

- Budgeted cost for work scheduled (BCWS)
- Budgeted cost for work performed (BCWP)
- Actual cost of work performed (ACWP)
- Budgeted cost at completion (BCAC)

For the Managed Level:

- BCWS, BCWP, and ACWP are determined for all activities on the project including (but not limited to):
  - Managing requirements
  - Software planning activities
  - Technical work
  - Subcontractor activities
  - Managing the subcontract
  - Software quality assurance activities
  - Software configuration management activities
  - Process definition and improvement activities
  - Intergroup coordination activities
  - Process measurement and analysis activities

- Defect prevention activities (for the Optimizing Level)
- Technology innovation activities (for the Optimizing Level)

Interpretation

The graphs and their interpretation are the same as those for the Defined Level.

The cost data for each project element allows the organization to determine cost of quality and to make informed decisions regarding tradeoffs. A Managed-Level organization has complete, accurate cost data for each activity on a project. For example, it knows how much it is spending on the collection and analysis of data, how much is spent on peer review activities and rework, and how much it saved by detecting defects early in the life cycle.

As part of the activities of a Managed-Level organization, quantitative product and process quality goals are established. When these quality goals conflict, (i.e., one goal cannot be achieved without compromising another), the project reviews and analyzes the software requirements, design, development plan and its quality plan, makes the necessary tradeoffs, and revises the quality goals appropriately. In order to make an informed decision, the costs for achieving the goals are part of software project planning. The cost information comes from the historical data. As part of the analysis, the project and the organization consider the customer and end users as well as the long-term business strategy and short-term priorities.

Sources

From the Defined Level:

[AFSC 86] has a discussion of these indicators.

[DoD 80] discusses the basics of ACWP, BCWP, and BCWS.

[DSDM 89] served as the major source of information in this section.

6.6. Quality

At the Managed Level, the quality indicators are divided among the results of software quality assurance audits, the results of life-cycle reviews with the customer, the trouble reports written after the implementation team has released the software for testing, and the results obtained from peer reviews.

6.6.1. Software Quality Assurance Audit Results

The software quality assurance audit results indicators for the Managed Level are the same as those at the Defined Level (see Section 5.5.1). A difference between these two levels is in the expectation of the numbers and types of noncompliance issues. At the Defined Level, a stable and well-defined software process exists. By the time an organization progresses to the Managed Level, the staff should be thoroughly familiar with the software process. Therefore, the organization should expect noncompliance issues resulting from process audits to be approaching zero. Similarly, the number of noncompliance issues can be expected to be less than at the Repeatable or Defined Levels. The number of product noncompliance issues may or may not approach zero; a reasonable goal can be obtained from the historical data.

6.6.2. Review Results

The review results indicators for the Managed Level are the same as those at the Defined Level (see Section 5.5.2).

6.6.3. Trouble Reports

At the Repeatable and Defined Levels, the number of trouble reports are used to denote quality. By itself, this is not a particularly objective quality indicator due to the different classes of people who write trouble reports (a consistency issue) and due to the range in severity of trouble reports. At the Managed Level, an organization can use other information besides trouble reports to determine the quality of the product and the process.

Objective of the Trouble Reports Indicator

To provide software managers with an insight into the quality of the product, the software reliability, and the effectiveness of testing and to provide the software engineering process group with information on the development processes.

Indicators

From the Defined Level:

- Trends in the following:
  - Number, type, and severity of the trouble reports
  - Trouble report density, that is, the number of trouble reports per unit size
  - Rate at which trouble reports are being addressed
  - Rate at which trouble reports are being written
  - Number of defects in each software component
- Relationship between the number of trouble reports and the number of test cases passed

For the Managed Level:

- Trends in the following:
  - Cause of the trouble reports
  - Testing, development, and implementation efficiency

Key Process Area Goals Addressed

Process Measurement and Analysis:

- Special causes of process variation (i.e., variations attributable to specific applications of the process, and not inherent in the process) are identified and controlled.

Quality Management:

- Measurable goals and priorities for product quality are established and maintained for each software project through interaction with the customer, end users, and project groups.

- Measurable goals for process quality are established for all groups involved in the software process.
- Process measurements are used to manage the software project quantitatively.

Life-Cycle Stages

- Integration and acceptance test
- Operations and maintenance

Users

From the Defined Level:

- Project software manager
- Software quality assurance personnel
- Software testing manager
- First-line and mid-level software development managers
- Software engineering process group

Users' Questions

From the Defined Level:

- Does the quality of the product indicate that the product is ready for release to the customer?
- Will undetected or unresolved problems in the product lead to more problems in the next life-cycle stage?
- Does the number of trouble reports indicate that the software product should be reworked before proceeding to the next life-cycle stage?
- Is the testing activity complete?
- Are project personnel addressing the trouble reports in a timely manner?
- Do the types of defects suggest areas for process changes?
- How does this project compare to others with regard to the number and types of defects discovered?
- Which components tend to be error prone?

For the Managed Level:

- Do the types of defects suggest areas for process changes?
- How does this project compare to others with regard to the number and types of defects discovered?
- How well is the project meeting its goals with respect to trouble reports?

Input

From the Defined Level:

- Number of trouble reports:
  - Total number
  - Open
  - Closed
- Number of test cases passed
- Product size
- For each trouble report:
  - Date opened
  - Date closed
  - Date trouble report evaluated
  - Type
  - Severity
  - Type (category) of defect
  - Severity of defect
  - Product identification
  - Trouble report identification
  - Source of problem
  - Cause of defect
  - Life-cycle stage in which trouble report is written
  - Activity in which defect was introduced
  - Units affected by defect

Interpretation

The interpretation of the indicators used at the Defined Level remains the same at the Managed Level, but the analysis of trouble reports focuses more on the analysis of the types of defects detected and testing efficiency.

Figure 6.6.3-1 shows the number of trouble reports for each type of defect and compares the numbers on this project to historical data. This figure is useful in determining what types of defects are generating the most trouble reports and suggests processes that may require improvement. The organization defines its own defect categories.

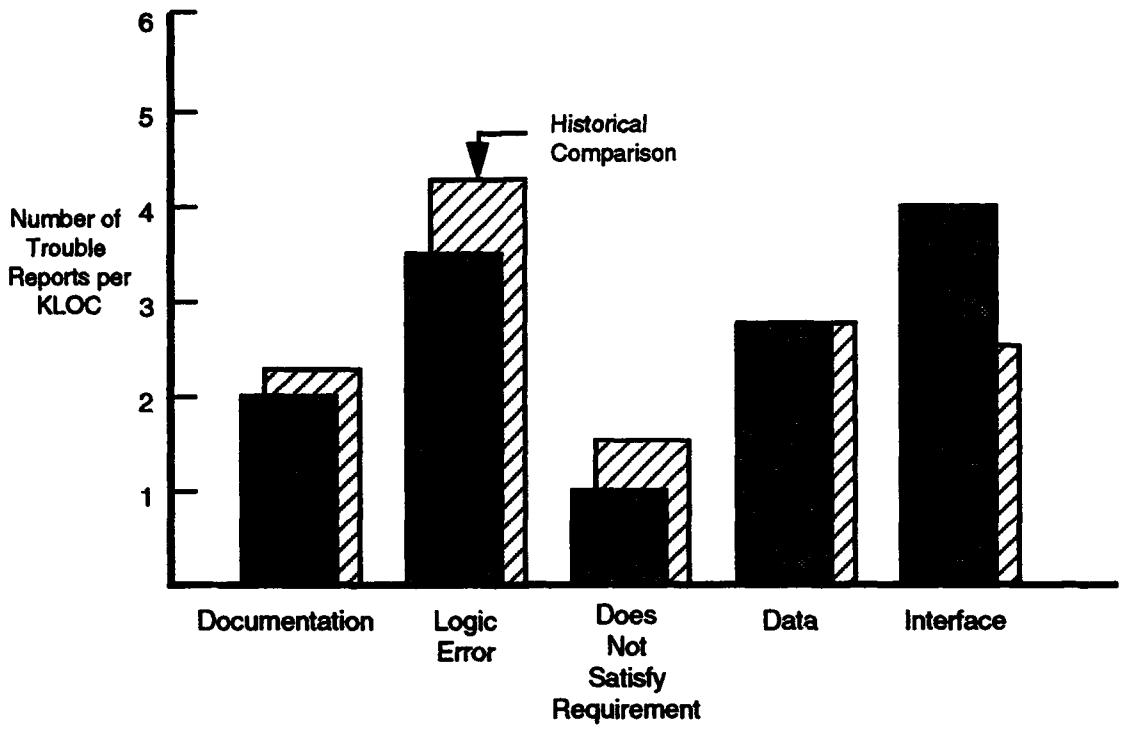


Figure 6.6.3-1. Number of Trouble Reports per Type of Defect

Once the defect type of each trouble report has been defined, the SEPG can determine the areas that are the sources of the trouble reports. Figure 6.6.3-2 shows the number of defects by type category [Mays 90].

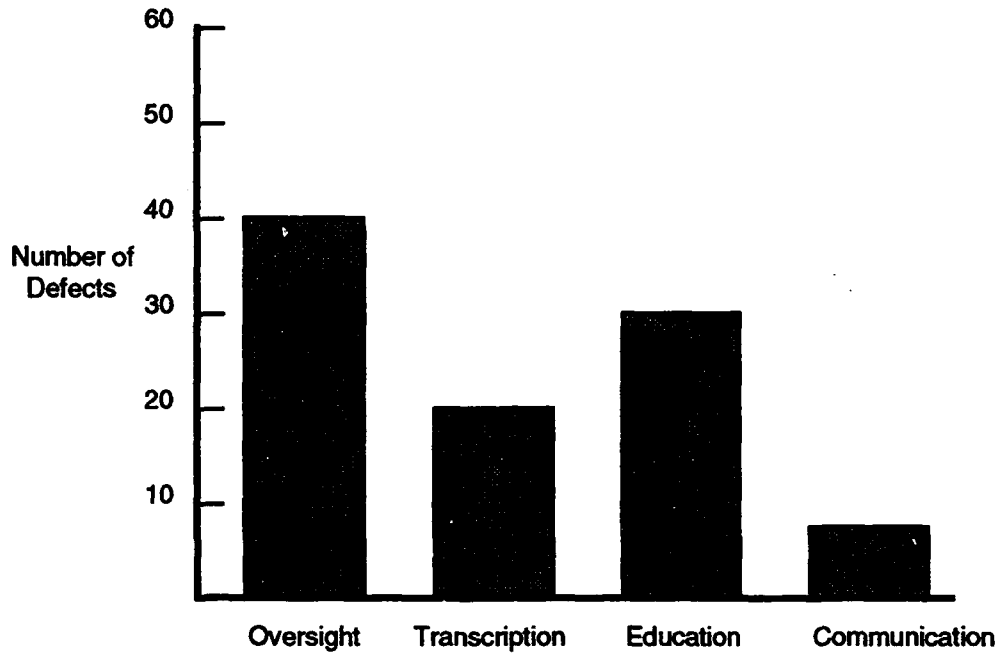


Figure 6.6.3-2. Number of Defects per Type Category

The figure has the following interpretation:

- A high number of defects due to oversight points out that a more rigorous review process may be necessary. More attention needs to be made to checking for consistency, completeness, and traceability.
- A high number of transcription defects points out that much more care needs to be taken when identifying variable names, etc. Simple errors in text transcription can be picked up by spell checkers, but variable names and so forth may require a rigorous review process. Further identification of the types of transcription defects experienced is necessary.
- A high number of defects due to education points out that some training may be needed to eliminate this type of problem. Further analysis is necessary to determine where the education is needed, such as in the process used to produce the product or in the skill level of the people developing the product.
- A high number of communication defects points out that the team is not working well together and the flow of information is not occurring efficiently. Analysis of where communication failures or bottlenecks are occurring is necessary. This may indicate need for training in communication techniques or a change in communication process.

At the Repeatable Level, the number of trouble reports generated were used to indicate the quality of the product. At the Managed Level, the number of defects recorded on the trouble reports provides efficiency indicators. Figure 6.6.3-3 shows three efficiency indicators for nine software releases of a project under statistical process control. The

delivered error rate for each release represents the development efficiency. It indicates the quality of the product released to the customer since it is a measure of the number of errors per thousand lines of code detected by the users during the warranty period or some set time period (e.g., the first six months after release). The implementation error rate represents the implementation efficiency. It indicates the quality of the products released to the independent test team since it measures the number of errors per thousand lines of code discovered by the test team. The test efficiency indicates the quality of the independent testing and is the ratio of the errors found by the independent test team to the total errors found by the test team and the customer during the warranty of set time period. The statistical control limits are also shown in the figure. If trouble report data during warranty or other set time period are not available from the customer, total error rate is the number of errors per thousand lines of code in each release delivered to the customer. The testing efficiency is the ratio of the errors found by the independent test team to the total number of errors found by both the independent test team and the acceptance test team. Figure 6.6.3-4 shows a similar figure for a project out of statistical control.

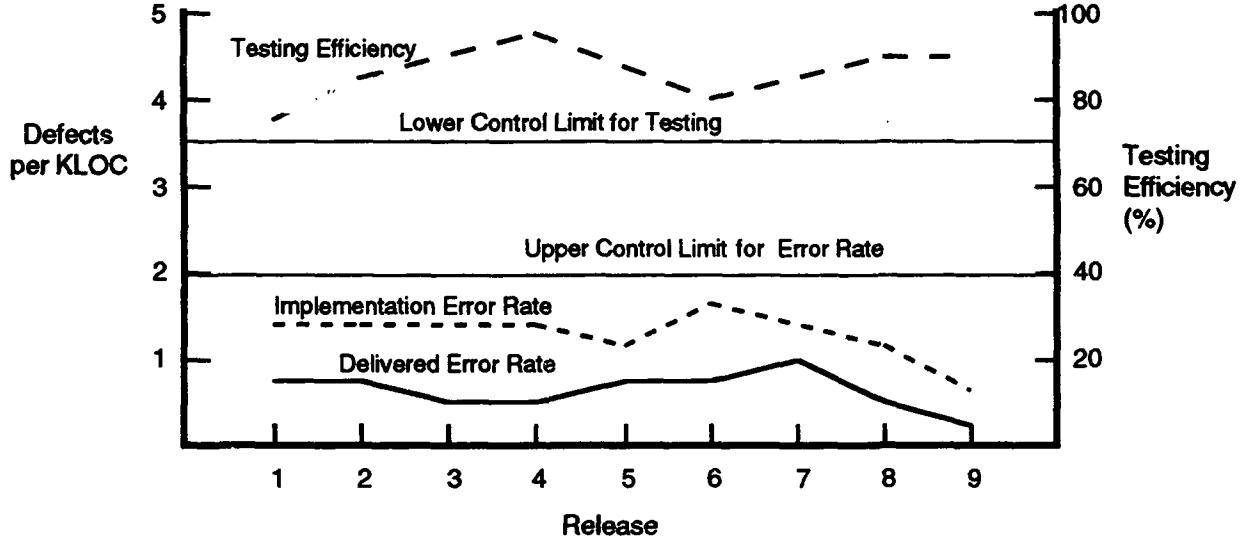


Figure 6.6.3-3. Efficiency Indicators for a Project in Statistical Control

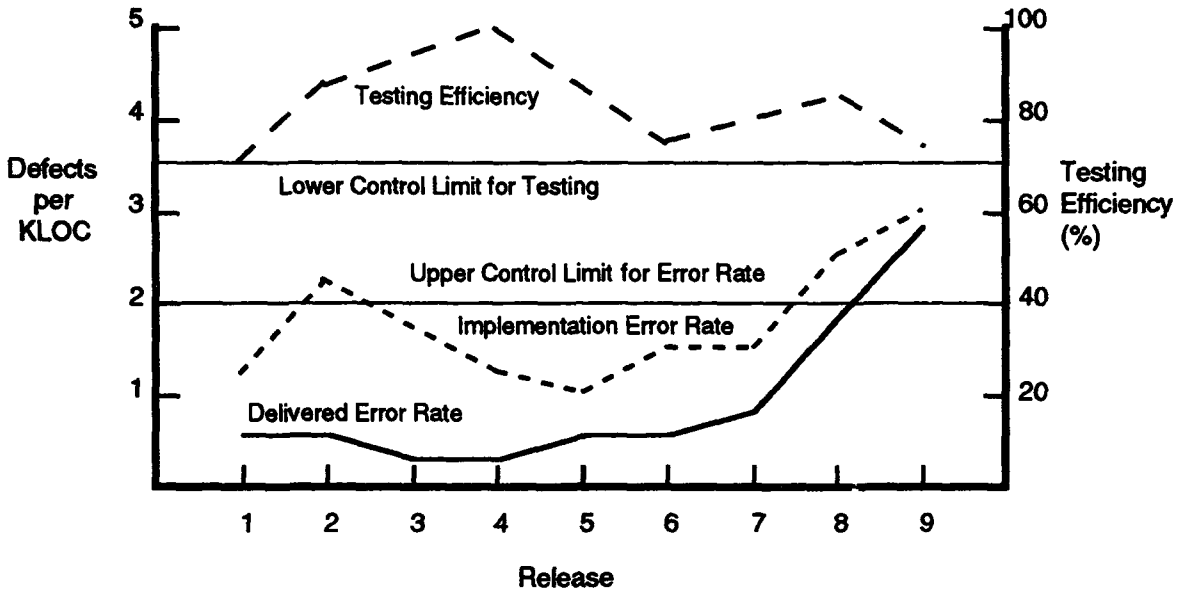


Figure 6.6.3-4. Efficiency Indicators for a Project out of Statistical Control

When interpreting efficiency indicators, the project software manger needs to remember to always look for process defects, not personnel defects. There may be a number of causes responsible for the defects. For example, an unrealistic implementation schedule or lack of appropriate resources can result in poor implementation efficiency, and inadequate time or resources for independent testing can result in poor testing efficiency.

Sources

From the Defined Level:

[AFSC 87], [Buckley 90], [Card 90], [Decker 91], [Grady 87], [IEEE 1061], [Landis 90], [Pfleeger 89], and [Rozum 92] discuss trouble reports.

[Florac 92] has a thorough discussion of trouble reports and serves as the main source of information for this section.

[Florac 92] has a thorough discussion of problem reports.

[IEEE 1044] was used to determine the inputs for this indicator.

For the Managed Level:

[DSDM 89] served as the source of information for the efficiency indicators.

[Mays 90] provided the categories in Figure 6.6.3-2.

6.6.4. Peer Review Results

Peer review results indicators are the same as those at the Defined Level. The difference between the two levels is in the use of statistical process control with the indicators, especially the number of defects detected during peer review and the number of re-reviews required. Additional analyses of defect data also occur, particularly in determining the characteristics of peer reviews.

Objectives of the Peer Review Results

- To provide the software managers and the software quality assurance personnel insight into the quality of the intermediate and final products.
- To provide the SEPG with insight into the peer review and development processes.

Indicators

From the Defined Level:

- Trends in the following:
  - Number of defects detected during peer reviews
  - Type and severity of defects
  - Number of re-reviews required
  - Rate at which defects are being addressed
  - Number of defects in each software component
- Number of defects in each product type
- Defect detection efficiency

Key Process Area Goals Addressed

Process Measurement and Analysis:

- The organization's standard software process is stable and under statistical process control.
- The relationship between product quality, productivity, and product development cycle time is understood in quantitative terms.
- Special causes of process variation (i.e., variations attributable to specific applications of the process and not inherent in the process) are identified and controlled.

Quality Management:

- Measurable goals and priorities for product quality are established and maintained for each software project through interaction with the customer, end users, and project groups.

- Measurable goals for process quality are established for all groups involved in the software process.
- Process measurements are used to manage the software project.

Life-Cycle Stages: All

Users

From the Defined Level:

- Software managers
- Software quality assurance personnel
- Software engineering process group

Users' Questions

From the Defined Level:

- Does the peer review indicate that the product quality is sufficient to allow the product to proceed to the next stage?
- Does the number of re-reviews suggest that there may be product quality and/or process problems?
- Do the types of defects detected during peer reviews suggest areas for process change?
- Do the peer reviews effectively detect defects?

Input

From the Defined Level:

- For each peer review:
  - Type of review (e.g., requirements review, design review, code review)
  - Number of items reviewed
  - Action items open
  - Action items closed
  - Identification of product reviewed
  - Product size
  - Preparation lead time
  - Preparation time for each reviewer
  - Length of time of review
  - Size of review team
  - Experience of review team
  - Structure of review team
  - Number of defects detected

- For each defect:
  - Severity
  - Type
  - Rework effort
  - Life-cycle stage in which it was introduced into product
  - Number of units affected by defect
  - Number of units containing defect
- Number of peer reviews
- Number of re-reviews

Interpretation

At the Managed Level, the organization continues to use the peer review indicators from the Defined Level, but now focuses on the characteristics of the peer review process. For example, each project can determine the following:

- The number KLOC reviewed per each hour in a review (KLOC/review-hour)
- The number KLOC reviewed per each hour of preparation (KLOC/preparation-hour)
- The number of hours it takes to detect a major defect (hours/major defect)
- The ratio of major to minor defects

A project can compare the values it has for these items and can compare them to the values for the organization as a whole and/or the project can compare the values it obtains for each release. For the latter, the project software manager can expect an improvement in each of the quantities since the project staff gain expertise both with the technical application and with the review process.

In Chapter 15, Humphrey has additional examples [Humphrey 89]. He shows plots of the number of errors per thousand lines of code (errors/KLOC) versus the review rate (KLOC/hour), review rate (KLOC/hour) versus preparation rate (KLOC/hour of preparation), and review rate (lines/hour) versus preparation time (hours/KLOC).

Figure 6.6.4-1 shows how defect detection efficiency varies with the inspection rate. The efficiency of each product review is plotted by its review rate. There is a curve fitted to the points that shows the overall view. This figure shows that as the review rate increases, the efficiency drops. It also shows that as the inspection rate decreases, the efficiency increases only to a point. Slower review rates are not cost effective. In this example, the lower defect detection efficiency for reviews that inspected source code at less than 75 LOC/hour may indicate that those review teams are not adequately trained in peer reviews. Preparation rate may also be used in addition to, or instead of, review rate.

A figure similar to Figure 6.6.4-1 can be used to plot defect detection rates (defects detected/hour) versus review rates. The same interpretation may be used.

Defect data also lends itself to statistical process control techniques since the sample is large enough, and the data are homogeneous (at least for a particular type of review). Humphrey also shows example control charts for the number of lines of code reviewed per hour, the number of defects detected per total review hours, the number of defects detected per thousand lines of code, and the number of hours of preparation per hour of review for each component of a project [Humphrey 89].

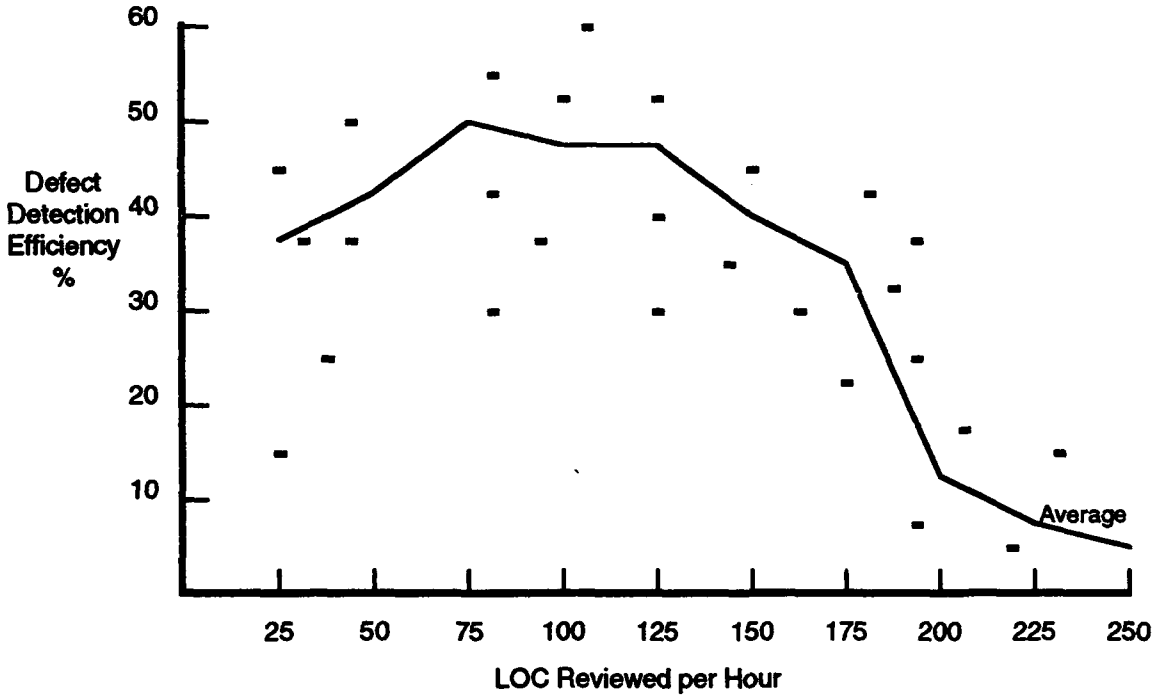


Figure 6.6.4-1. Defect Detection Efficiency vs. Review Rate

Sources

From the Defined Level:

[Decker 91] and [Florac 92] were the major sources of information for this section.

[AFSC 87], [Buckley 90], [Card 90], [Grady 87], [IEEE 1061], [Landis 90], [Pfleeger 89], and [Rozum 92] discuss concepts on tracking and analyzing trouble reports that are applicable to peer review data.

[IEEE 1044] was consulted for the types of inspection data to be collected.

[Pyzdek 89] discusses Pareto analysis.

For the Managed Level:

[Humphrey 89] discusses measurement of the peer review process.

6.7. Stability

At the Managed Level the stability indicators concentrate on the stability of the requirements, size, and process. Requirements stability is concerned with the number of changes to the requirements, the number of waivers from requirements, and the length of time the requests for changes to requirements are open. Size stability is concerned with code size and size estimates. Process stability is concerned with the number of changes to the software process and the number of waivers from the process.

6.7.1. Requirements Stability

The requirements stability indicators for the Managed Level are the same as those at the Defined Level (see Section 5.6.1).

6.7.2. Size Stability

The size indicators for the Managed Level are the same as those at the Defined Level. At the Managed Level, the manager can apply statistical process control to the historical data from other similar projects. Any variation from the plan that is still within the upper and lower statistical limits is considered normal. Variations outside this normal range require analysis.

Objective of the Size Stability Indicator

To provide the project software manager and the project manager with an indication of the completeness and stability of the requirements and of the capability of the implementation staff to produce the software product within the current budget and schedule.

Indicators

Same as the Repeatable Level:

- Trends in the code size
- The variation of actual software size from size estimates
- Variation of actual software size from estimated size by build or release

Key Process Area Goals Addressed

Process Measurement and Analysis:

- The relationship between product quality, productivity, and product development cycle time is understood in quantitative terms.

Quality Management:

- The software plans, design, and process are adjusted to bring forecasted process and product quality in line with the goals.
- Process measurements are used to manage the software project quantitatively.

Life-Cycle Stages: All

Users

Software engineering and project software managers for controlling and monitoring of project

Users' Questions

From the Defined Level:

- How much have the size estimates changed over time during development?
- How much do the actual values deviate from their estimated values?

- How much does the trend of the actual values affect the development schedule?
- Is the estimated productivity sufficient to allow the completion of added code on schedule or are more staff required?

For the Managed Level:

- Are the variations in the size estimates becoming smaller over time?
- Are the variations in the size estimates within the control limits?

Input

From the Defined Level:

- Software size estimates:
  - Total size
  - By computer software configuration item (CSCI)
  - New code
  - Off-the-shelf code
  - Reused code
- Amount of software scheduled for completion by build or release

Interpretation

Figure 5.6.2-1 shows the change in software size over time for the project and in comparison with similar projects. Figure 6.7.2-1 shows a similar figure. The planned line represents the planned software size at various times in the development life cycle. The actual line represents the actual software size on those dates. The upper and lower control limits along the Now line are derived from historical data for similar projects and show whether the difference between the planned and actual lines is part of the normal variation. The interpretation of the figure remains the same as for Figure 5.6.2-1.

Figure 6.7.2-2 shows a similar figure. In this example, the planned size estimate has remained constant since month 5. The actual software size has also stayed within the statistical limits. The differences fluctuate about the planned line. No immediate action is required by the project manager. The manager, however, may wish to establish a goal of minimizing the variation or to at least have the difference between the planned and actual lines constant since the fluctuations are an indication of the amount of rework that is occurring.

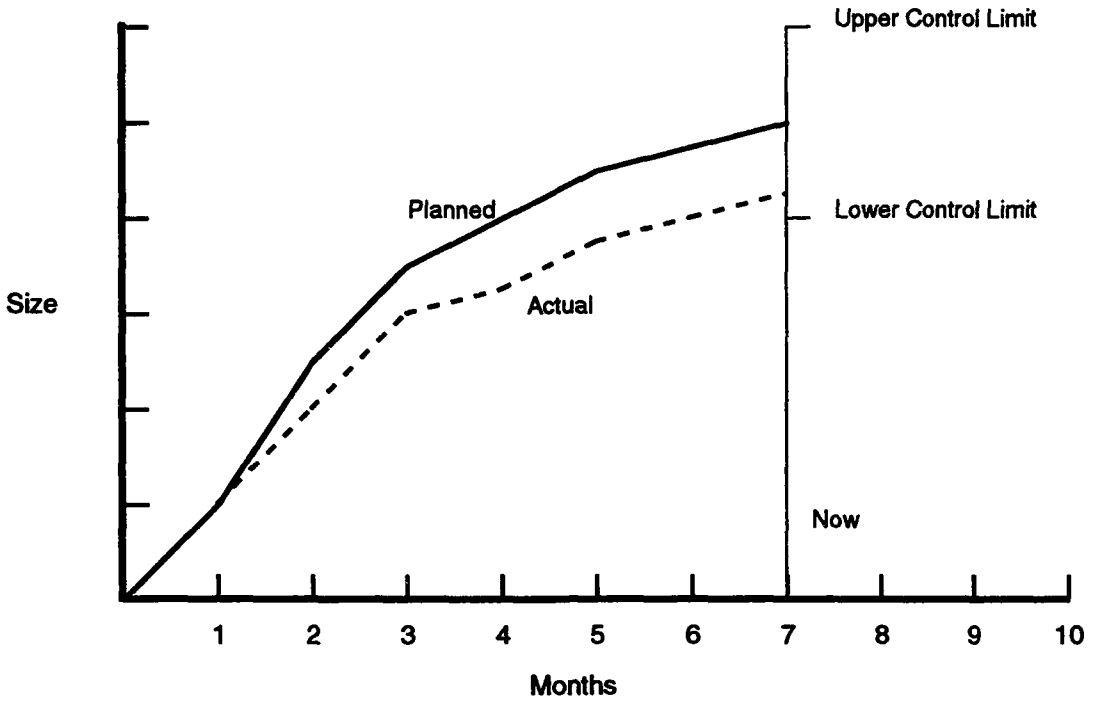


Figure 6.7.2-1. Software Size as a Function of Time Showing Upper and Lower Control Limits

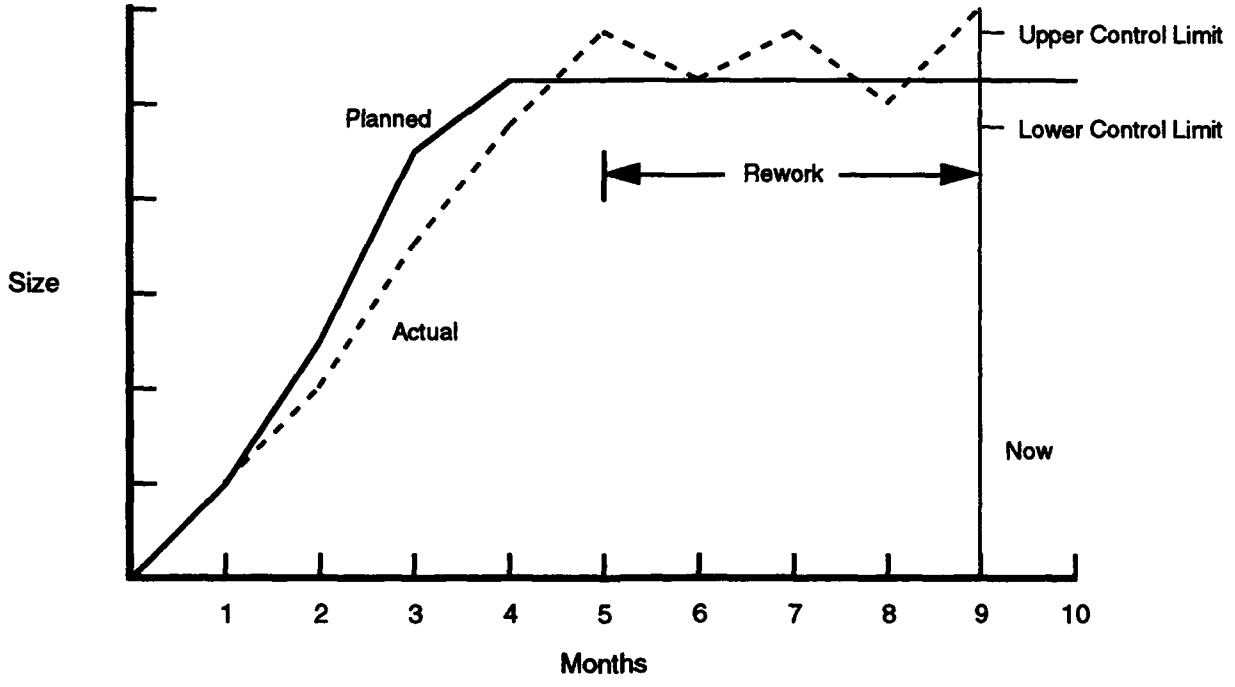


Figure 6.7.2-2. Software Size as a Function of Time

Sources

From the Repeatable Level:

[AFSC 86], [Decker 91], [Landis 90], [Pfleeger 89], and [Schultz 88] all discuss tracking software size.

6.7.3. Process Stability

The process stability indicators are the same as the Defined Level (see Section 5.6.3).

6.8. Computer Resource Utilization

The computer resource utilization indicators are the same as the Repeatable Level (see Section 4.7.).

6.9. Training

The training indicators are the same as the Defined Level (see Section 5.8).

7. The Optimizing Level—Maturity Level 5

This chapter summarizes the characteristics of an organization with an optimizing process and discusses the indicators that are appropriate for the Optimizing Level.

7.1. Characteristics of an Optimizing-Level Organization

An organization with an optimizing process is characterized as one that has the means to identify weak process elements and strengthen them, with the goal of preventing the occurrence of defects. Statistical evidence is available on process effectiveness and is used in performing cost benefit analyses on new technologies. Innovations that exploit the best software engineering practices are identified.

Indicators appropriate for the Optimizing Level are progress, effort, cost, quality, stability, computer resource utilization, and training. As noted in Chapter 1, there are few examples of proven indicators for this maturity level. The discussion of indicators in this chapter is based on expected practice. As organizations mature, experience with indicators at this maturity level will increase, and the information in this chapter could change.

7.2. Progress

At the Managed Level, progress is controlled and statistically measured. The progress indicators at the Optimizing Level examine the effect of process improvement activities, technology innovation, and defect prevention activities on progress. Changes to the development process should cause time spent in activities to decrease (increased productivity). While changes to processes occur, progress is not under statistical control, but will stabilize after a period of time. The new progress rates should be higher, and the variation in performance should be tighter. That is, the control limits will be closer to the average. While the processes are changing, progress should be monitored closely to ensure that the changes are not detrimental.

Objective of the Progress Indicators

To provide managers with information on the effects of defect prevention, technology innovation, and process improvement on projects' progress.

Indicators

- Ratio of rework time to total project time per project
- Trend in the rate of time spent in activities undergoing process change

Key Process Area Goals Addressed

Defect Prevention

- Sources of product defects that are inherent or repeatedly occur in the software process activities are identified and eliminated.

Technology Innovation

- Selection and transfer of new technology into the organization is orderly and thorough.

Process Change Management

- The organization's standard software process and the projects' defined software processes continually improve.

Life-Cycle Stages: All

Users

- All levels of project software management
- Software Engineering Process Group (SEPG)

Users' Questions

To what extent have defect prevention, technology innovation, and process improvement shortened total development time?

Input

- Project software development plan
- Planned and actual time spent in activities

Interpretation

The use of defect causal analysis and defect prevention procedures can reduce the time spent in rework activities. Likewise, improvements made by changes to the development processes and technical innovation can reduce the time spent in other activities. This reduces total development time (time to market). The SEPG tracks the ratio of time spent in changed activities to the total project time for all projects. Figure 7.2-1 shows a scatter diagram of the rework ratios for several projects.

In Figure 7.2-1, the y-axis is the ratio of time spent in rework to total project time. Each point in the diagram is a project that completed in that month. Projects that started before the defect prevention procedures were enacted and were completed afterwards show only a partial drop in the ratio. Projects started after the procedures were enacted have lower ratios and tend to group together. Once the amount of rework is reduced, schedules can correspondingly be shortened as the productivity has improved.

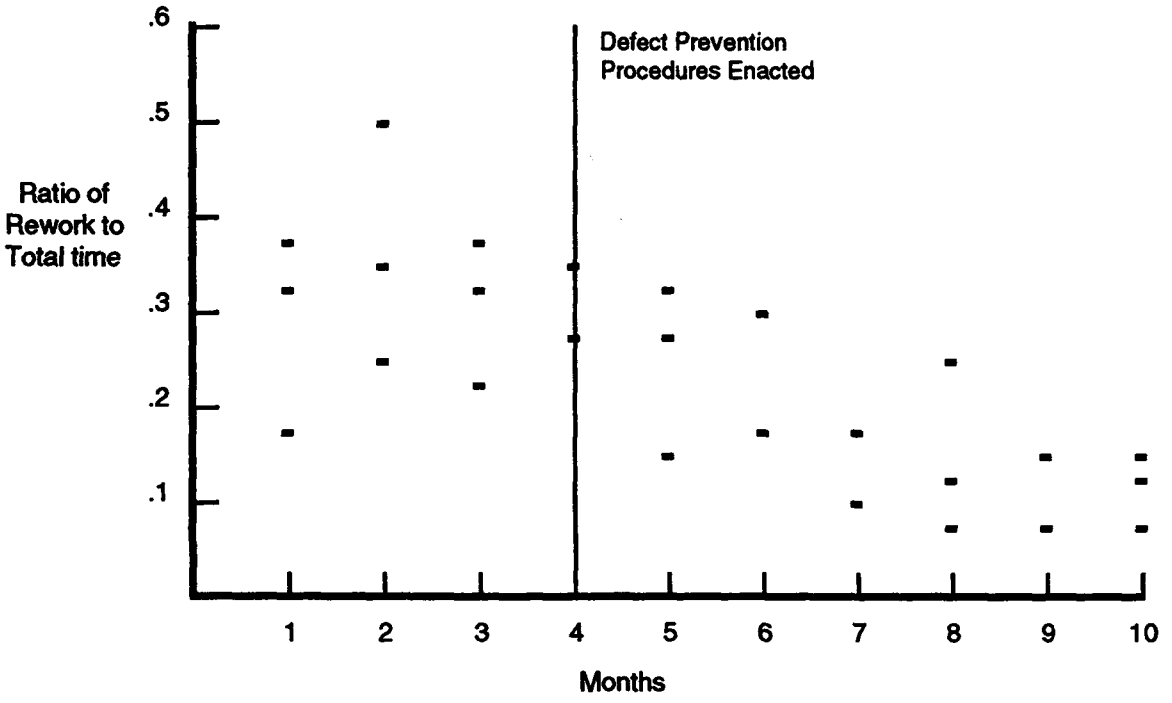


Figure 7.2-1. A Scatter Diagram of the Ratio of Time Spent in Rework Activities for Several Projects

Figure 7.2-2 shows the same diagram with control limits and an average line derived from the data. Before month 4, the ratios are high and vary widely. Defect prevention procedures are enacted during month 5. Although there is still large variation during month 5 through month 7, the trend indicates that the procedures are effective. By month 8, the new procedures have stabilized, the ratios have dropped to a steady state and new control limits are established.

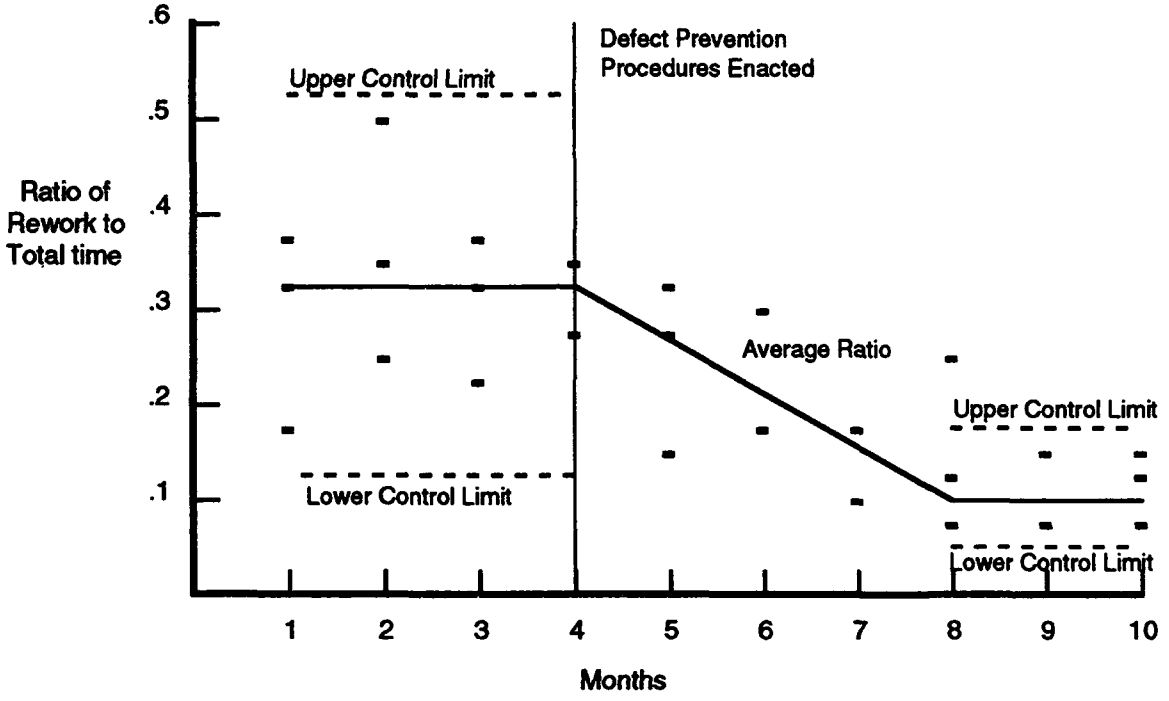


Figure 7.2-2. The Ratio of Time Spent in Rework Activities for Several Projects with Control Limits

This figure can also be used to track the ratios of peer reviews, coding, testing, and other activities to the total development time. In turn, this allows an organization to determine its performance with respect to preventive and reactive efforts, that is, the organization can compare the time spent in design as opposed to coding or the time spent in peer reviews to testing. The emphasis should be on preventive activities.

Sources

From the Repeatable Level:

[AFSC 86] discusses planned and actual completions graphs at the computer software configuration item (CSCI) level.

[Decker 91] lists requirements diagrams; function specifications; design diagrams; test cases; units designed, coded, and tested; modules tested; and computer software components tested as items tracked on planned and actual completions graphs.

[Grady 87] states that calendar measures are part of the Hewlett-Packard metrics program.

[Landis 90] discusses planned and actual completions graphs for units coded, read, and tested.

[Rozum 92] has a discussion on Gantt charts in their milestone performance metric and a discussion of planned and actuals in their development progress metric.

[Schultz 88] discusses software requirements documented in his design progress metric and the number of computer software units (CSU) designed, coded, tested, and integrated in his CSU development progress metric. He also discusses the planned and actual completions of CSCIs integrated in his test progress metric.

[STEP 91] discusses a schedule metric upon which Figure 4.2-2 is based and a development progress metric.

From the Defined Level:

[Lockyer 84] discusses Gantt charts and the critical path method.

For the Optimizing Level:

[Pyzdek 89] discusses SPC techniques.

7.3. Effort

The effort indicators at the Optimizing Level examine the effects of process improvement activities, technology innovation, and defect prevention activities on effort. Changes to the development process should cause effort spent in activities to decrease.

Objective of the Effort Indicators

To provide managers with information on the effects of defect prevention, technology innovation, and process improvement on projects' effort.

Indicators

- Ratio of rework effort to total project effort per project
- Trend in the rate of effort spent in activities undergoing process change

Key Process Area Goals Addressed

Defect Prevention:

- Sources of product defects that are inherent or repeatedly occur in the software process activities are identified and eliminated.

Technology Innovation:

- Selection and transfer of new technology into the organization is orderly and thorough.
- Technology innovations are tied to quality and productivity improvements of the organization's standard software process.

Process Change Management:

- The organization's standard software process and the projects' defined software processes continually improve.

Life-Cycle Stages: All

Users

- All levels of project software management
- Software Engineering Process Group (SEPG)

Users' Questions

To what extent have defect prevention, technology innovation, and process improvement reduced effort?

Input

- Project software development plan
- Planned and actual effort spent in activities

Interpretation

The use of defect causal analysis and defect prevention procedures can reduce the effort spent in rework activities. Likewise, improvements made by changes to the development processes and technical innovation can reduce the effort spent in other activities. This reduces total development effort (improved productivity). The SEPG tracks the ratio of effort spent in changed activities to the total project effort for all projects. Figure 7.3-1 shows a scatter diagram of the rework ratios for several projects with control limits and an average line derived from the data.

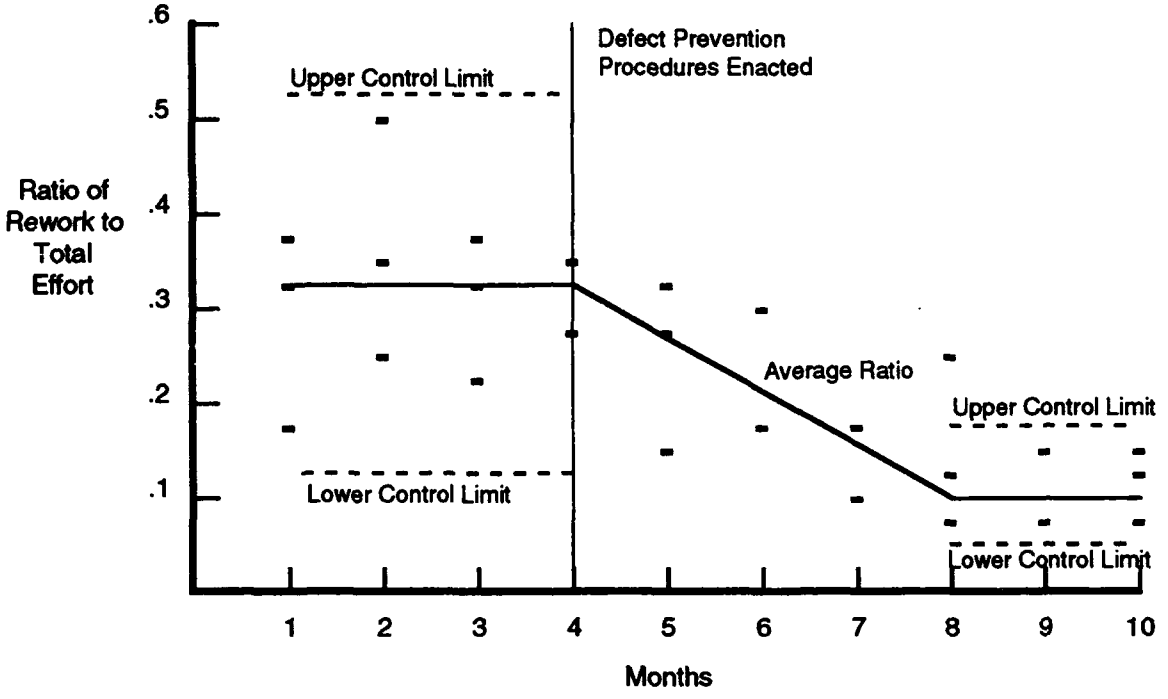


Figure 7.3-1. The Ratio of Effort Spent in Rework Activities for Several Projects

In Figure 7.3-1, the y-axis is the ratio of effort spent in rework to total project effort. Each point in the diagram is a project that completed in that month. Projects that started before the defect prevention procedures were enacted and were completed afterwards show only a partial drop in the ratio. Projects started after the procedures were enacted have lower ratios and tend to be clumped together. Once the amount of rework is reduced, productivity improves since the staff does not have to repeat work in a particular activity. Fewer iterations are required to complete the activity or product.

In the figure, the ratios are high and vary widely before month 4. Defect prevention procedures are enacted during month 5. Although there is still large variation during month 5 through month 7, the trend indicates that the procedures are effective. After

month 8, the ratios have dropped to a steady state and new control limits are established.

This figure can also be used to track the ratios of the effort spent in peer reviews, coding, testing, and other activities to the total development effort. In turn, this allows an organization to determine its performance with respect to preventive and reactive efforts; that is, the organization can compare the effort expended in design as opposed to coding or the effort expended in peer reviews to testing. The emphasis should be on the preventive activities.

Sources

From the Repeatable Level:

[AFSC 86] discusses planned and actual staffing total profiles and staffing losses in its software development personnel indicator.

[Decker 91], [Landis 90], and [Pfleeger 89] discuss the use of planned and actual staffing profiles.

[Grady 87] reports that staff issues are part of the Hewlett-Packard software metrics program.

[IEEE 1045] discusses the experience level, size, and turnover rates of the project staff.

[Rozum 92] discusses planned and actual staffing total profiles, experience profiles, and also planned and actual staffing losses in their effort and staffing metrics.

[Schultz 88] discusses planned and actual staffing total profiles, experience profiles, and also planned and actual staffing losses in his software personnel metric.

For the Optimizing Level:

[Pyzdek 89] discusses SPC techniques.

7.4. Cost

At the Managed Level, the project software manager has a thorough understanding of the detailed costs on the project. At the Optimizing Level, the project software manager can take advantage of this knowledge to conduct cost/benefit analyses whenever there is the need or desire to insert a new technology or introduce a process modification.

Objective of the Cost Indicators

To provide management with information that supports cost/benefit analyses for making process improvement, technological innovation, and defect prevention planning decisions.

Indicators

- Comparative costs and benefits of alternative process improvement activities, defect prevention activities, or technologies.
- Planned versus actual cost and benefit of an alternative process improvement activity, defect prevention activity, or technology.

Key Process Area Goals Addressed

Technology Innovation:

- The organization has a software process and technology capability to allow it to develop or capitalize on the best available technologies in the industry.
- Selection and transfer of new technology into the organization is orderly and thorough.

Life-Cycle Stages: All

Users: All levels of management

Users' Questions

- Has the cost estimation process improved so that the cost and schedule variances are approaching zero?
- Do the variances indicate areas that need process improvement?
- Which technology from the alternatives is best suited to address problem areas?
- Is the process improvement activity, or defect prevention activity, or technology producing the desired results?

Input

From the Repeatable Level:

- Budgeted cost for work scheduled (BCWS)
- Budgeted cost for work performed (BCWP)
- Actual cost of work performed (ACWP)

- Budgeted cost at completion (BCAC)

From the Managed Level:

- BCWS, BCWP, and ACWP are determined for all activities on the project including (but not limited to):
  - Managing requirements
  - Software planning activities
  - Technical work
  - Subcontractor activities
  - Managing the subcontract
  - Software quality assurance activities
  - Software configuration management activities
  - Process definition and improvement activities
  - Intergroup coordination activities
  - Process measurement and analysis activities
  - Defect prevention activities (for the Optimizing Level)
  - Technology innovation activities (for the Optimizing Level)

For the Optimizing Level:

- Planned and actual cost and benefit for each alternative process improvement activity, defect prevention activity, or technology.

Interpretation

One goal of an organization is to continuously improve its estimating process. The cost and schedule variances formed from the BCWS, BCWP, and ACWP allow the organization to determine the efficiency of its cost estimation process as shown in Figure 7.4-1.

Figure 7.4-1 is a plot of the final cost variances for projects within the organization. As time progresses and the cost estimation process improves, the variances decrease. This shows that the process improvement that was implemented is effective since the estimates are becoming more reliable and accurate.

An analysis of the cost and schedule variances at lower levels within the work breakdown structure can indicate areas that need improvement. If an activity routinely has a larger variance than other activities on the project or on several projects, the organization can identify this as an activity that needs a process change or may benefit from a different method or technology.

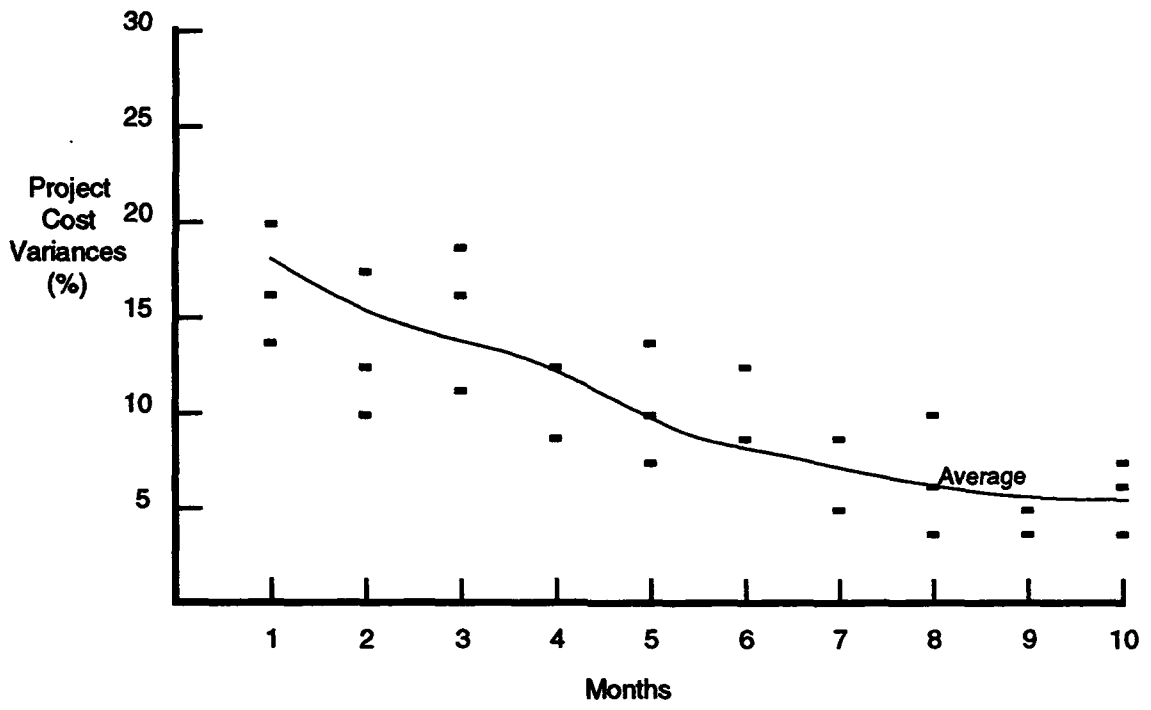


Figure 7.4-1. Project Cost Variances

To improve a process or use a new technology, the organization first selects alternative process improvement activities, defect prevention activities, or technologies that may improve the software development process. These activities and technology alternatives must be selected, in part, on the basis of their cost and benefit. The cost is an estimate of the amount of investment that must be made to get a return (benefit). The costs and benefits are estimates based on total costs and benefits over a predetermined period of time. Figure 7.4-2 shows the cost and the benefit for several alternative activities that are not necessarily mutually exclusive.

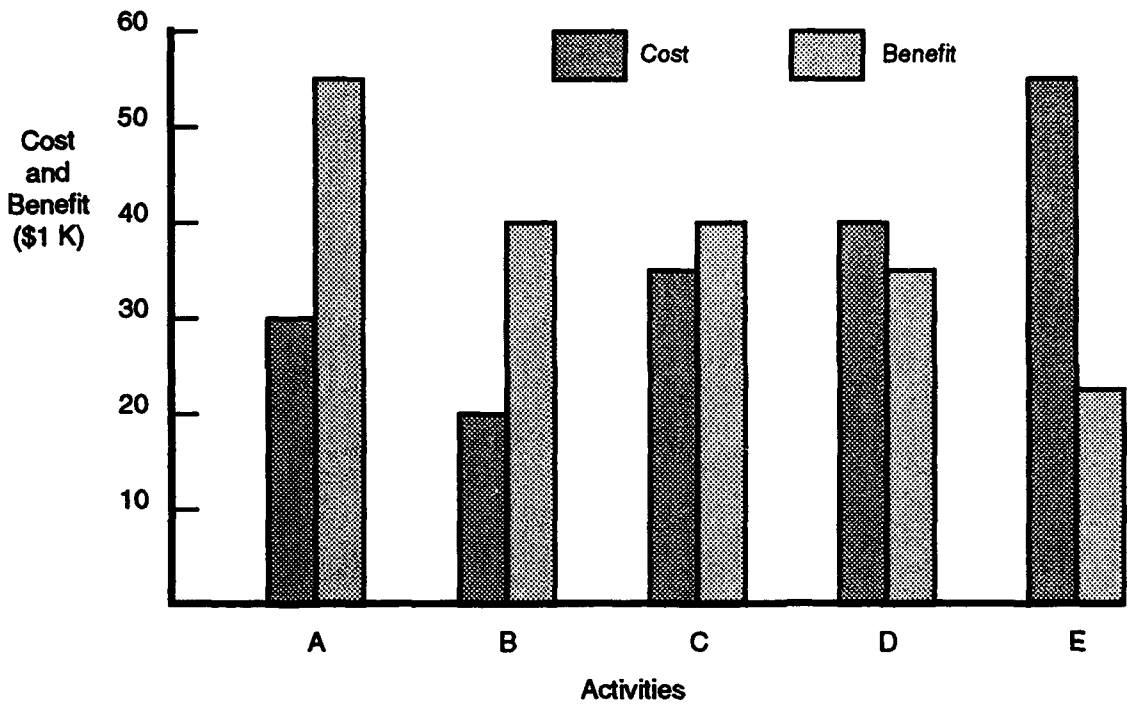


Figure 7.4-2. Cost and Benefit Trade-Offs of Improvement Activities

The activities in Figure 7.4-2 are ordered from left to right by the size of the difference between their cost and benefit. Activity A has a greater cost and a higher cost/benefit ratio than activity B, but A has a greater, positive difference between cost and benefit. That is, A has a greater net benefit. Since the costs and benefits are estimates, activities C and D could be considered equivalent. In addition to the analysis in the figure, the organization must consider, at a minimum, the difficulty in implementing the various alternatives, the readiness of the organization for the change, and the length of time to implement the change before it makes its final decision.

Once an activity, technology, tool, or method has been selected and implemented, the actual cost and benefits must be tracked against the original estimates. Figure 7.4-3 shows the estimated and actual cumulative cost and benefits for peer review training.

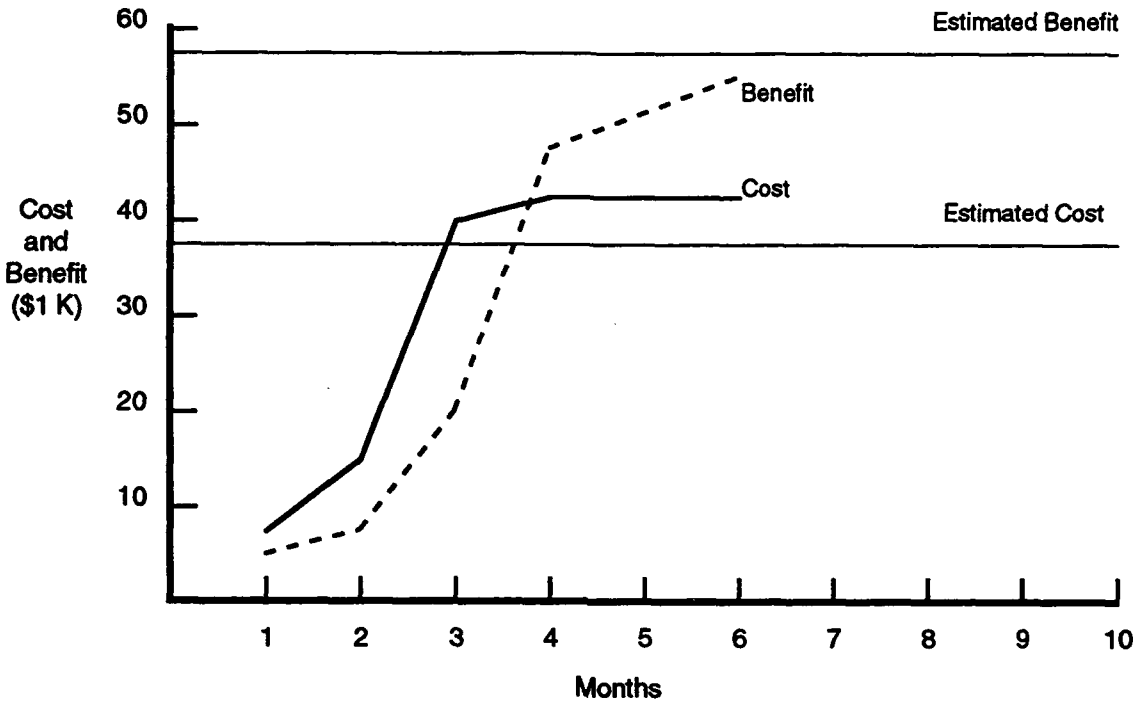


Figure 7.4-3. Cost and Benefits of Peer Review Training

The estimated cost and benefit lines in Figure 7.4-3 represent the total cost and benefit over time. In this example, the cost of the training has already exceeded the original estimate but is no longer growing. The benefit due to early, lower-cost defect removal has not yet reached the original estimate but is approaching it.

Sources

From the Repeatable Level:

[AFSC 86] and [DoD 80] discuss the basics of ACWP, BCWP, and BCWS.

[DSDM 89] discusses the basics of cost and schedule variance.

7.5. Quality

At the Optimizing Level, quality indicators are divided among the results of software quality assurance audits, the results of life-cycle reviews with the customer, the trouble reports written after the implementation team has released the software for testing, the results obtained from peer reviews, and defect prevention.

7.5.1. Software Quality Assurance Audit Results

The software quality assurance audit results indicators for the Optimizing Level are the same as the Defined and Managed Levels (see Section 4.5.1 and Section 5.5.1).

7.5.2. Review Results

The review results indicators for the Optimizing Level are the same as those for the Defined Level (see Section 5.5.2).

7.5.3. Trouble Reports

The trouble reports indicators for the Optimizing Level are the same as those for the Managed Level (see Section 5.5.3).

7.5.4. Peer Review Results

The peer review results indicators for the Optimizing Level are the same as those for the Managed Level (see Section 5.5.4).

7.5.5. Defect Prevention

Defect prevention should be a goal for every project member. If zero defects is not achievable, an alternate goal is to eliminate those defects that occur repeatedly. The defect prevention indicators enable the project management, the software engineering process group (SEPG), and software quality assurance personnel to determine the most common defects on projects and provide the capability to determine whether their defect prevention processes are effective.

Objective of the Defect Prevention Indicators

To provide software managers, SEPG, and software quality assurance personnel insight into the cause of defects and the effect of defect prevention activities on defection insertion rates.

Indicators

- Defect category profiles
- Trend in defect insertion rate

Key Process Area Goals Addressed

Defect Prevention:

- Sources of product defects that are inherent or repeatedly occur in the software process activities are identified and eliminated.

Life-Cycle Stages: All

Users

- Software managers
- Software engineering process group
- Software quality assurance personnel

Users' Questions

- What types of defects are being inserted?
- Which types of defects are being inserted the most?
- What is the effect on the defect insertion rate when defect prevention procedures are enacted?
- Are defect prevention procedures actually preventing defects?

Input

- Trouble reports
- Size and effort (for normalizing the defect data)

Interpretation

Table 7.5.5-1 shows where defects are being inserted and detected by life-cycle activities. The rows of the table list the development activities where defects were detected. The columns list the activities where defects were inserted. In this example, the spread is what would be expected except for the high number of requirements and preliminary-design defects that are not caught until the acceptance test. The table can also be used for specific categories of defects.

Inserted / Detected	Requirements	Preliminary Design	Critical Design	Code
Requirements	43			
Preliminary Design	27	32		
Critical Design	14	17	55	
Code	3	7	26	73
Unit Test	3	4	17	58
Integration Test	4	5	4	18
Acceptance Test	12	10	2	12

Table 7.5.5-1. Defect Insertion and Detection by Life-Cycle Activity

A variation of Table 7.5.5-1 is Figure 7.5.5-2 which shows a horizontal or vertical slice of the table. The figure can show where defects are inserted and how long it takes to detect them, or it can show where the defects detected in an activity originated. This example shows the percentage of defects inserted in the preliminary design activity that are detected in design and subsequent activities. The raw number of defects can also be used.

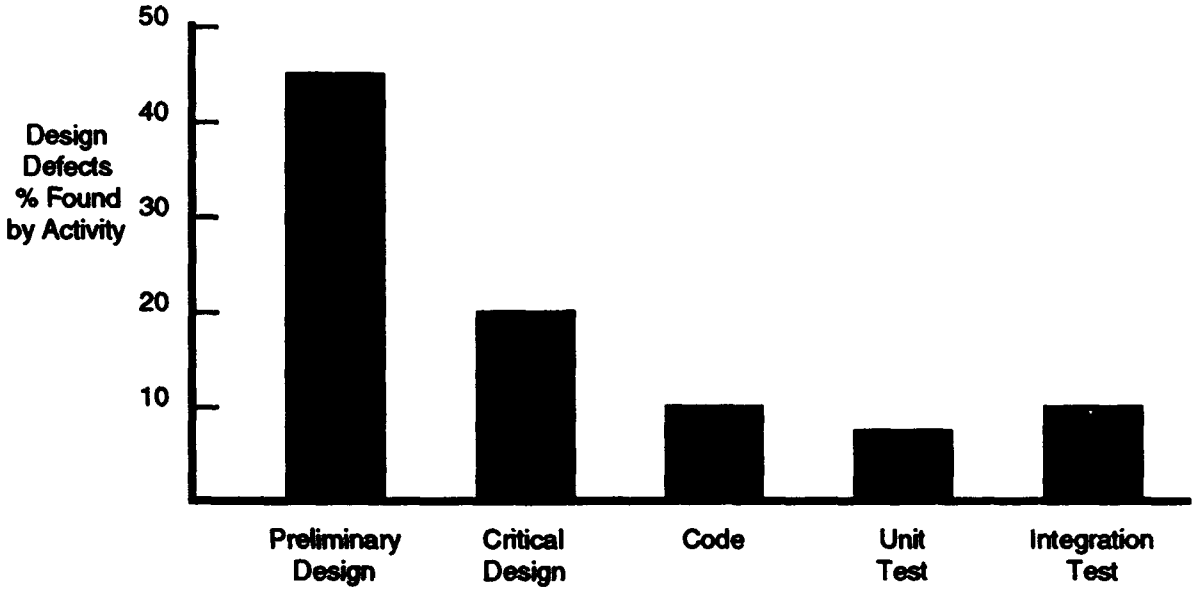


Figure 7.5.5-2. Design Defects Detected by Life-Cycle Activity

Figure 7.5.5-3 is a histogram of the number of defects detected by category. The figure can also use percentages to allow inter-project comparisons. Pareto analysis can be performed to find which types of defects are most prevalent.

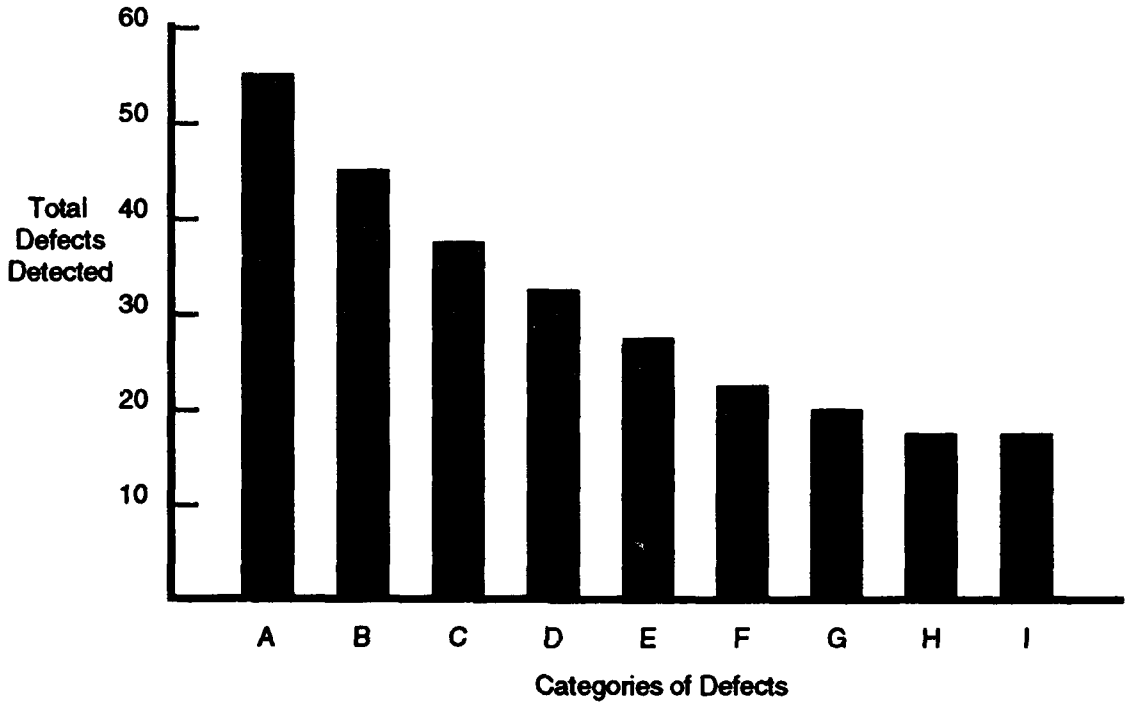


Figure 7.5.5-3. Histogram of Categories of Defects Detected

Some defects are more costly to remove than others; therefore, a category with relatively few defects may represent a greater cost than other categories with more defects. A variation of Figure 7.5.5-3 can be used to show the total cost of defects for all defects in a category. This information can be used to select appropriate defect prevention procedures.

Figure 7.5.5-4 addresses the defect prevention process. It shows what happens to a particular defect category when defect prevention procedures are enacted. The x-axis is time or stages. The y-axis is the raw number of defects inserted or the number of defects inserted normalized by size. In this example, prior to month 4, there is a high insertion rate which varies widely from month to month for type A defects. After the defect prevention procedures were enacted, the insertion rate dropped. By month 7 the rate stabilized and new, narrower control limits were placed on the rate.

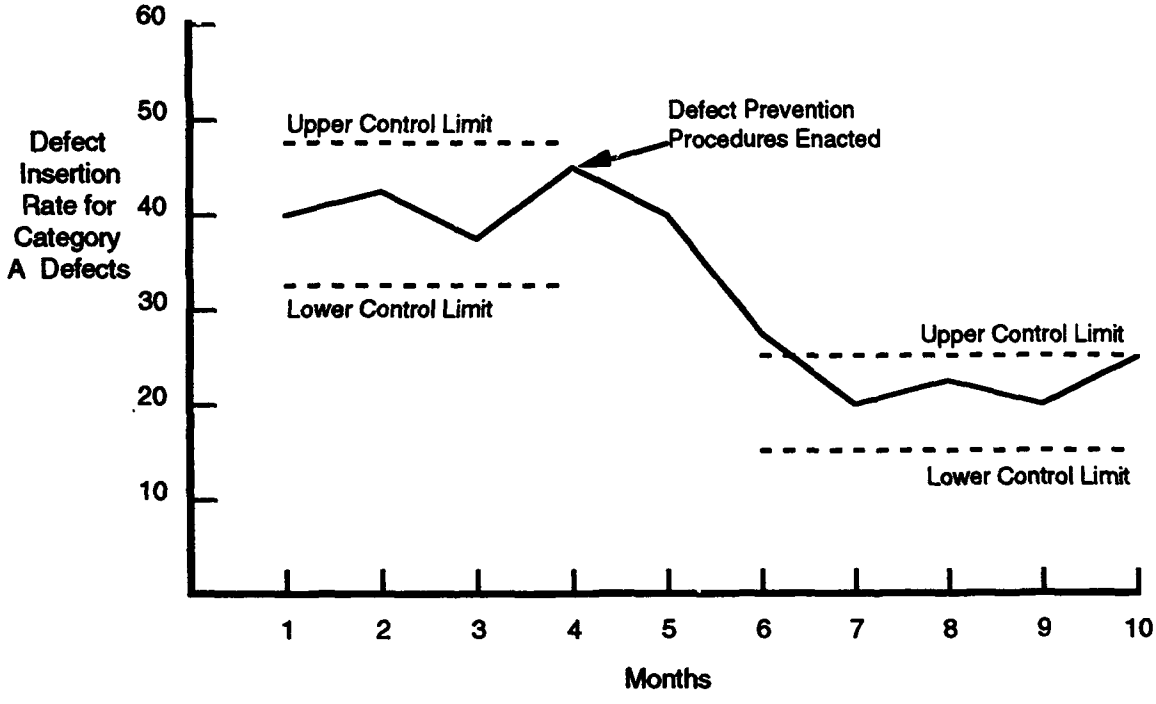


Figure 7.5.5-4. Defect Insertion Rate for Category A Defects

Sources

[Mays 90] discusses defect prevention.

7.6. Stability

The stability indicators for the Optimizing Level are the same as those for the Defined and Managed Levels (see Sections 5.6 and 6.7).

7.7. Computer Resource Utilization

The computer resource utilization indicators for the Optimizing Level are the same as those for the Repeatable Level (see Section 4.7).

7.8. Training

The training indicators for the Optimizing Level are the same as those for the Defined Level (see Section 5.8).

References

[AFSC 86] Department of the Air Force. *Air Force Systems Command Software Management Indicators* (AFSC Pamphlet 800-43). Washington, DC: Andrews Air Force Base, 1986.

[AFSC 87] Department of the Air Force. *Air Force Systems Command Software Quality Indicators* (AFSC Pamphlet 800-14). Washington, DC: Andrews Air Force Base, 1987.

[Basili 84] Basili, Victor R.; & Weiss, David M. "A Methodology for Collecting Valid Software Engineering Data." *IEEE Transactions on Software Engineering SE-10,* 6 (November 1984): 728-738.

[Brooks 82] Brooks, F. P. Jr. *The Mythical Man-Month: Essays on Software Engineering.* Reading, MA: Addison-Wesley Publishing Co., 1982.

[Buckley 89] Buckley, Fletcher J. "Standard Set of Useful Software Metrics is Urgently Needed." *IEEE Computer 22,* 7 (July 1989): 88-89.

[Buckley 90] Buckley, Fletcher J. "Establishing a Standard Metrics Program." *IEEE Computer 23,* 6 (June 1990): 85-86.

[Card 88] Card, David N. "Software Product Assurance: Measurement and Control." *Information and Software Technology 30,* 6 (July/August 1988): 322-330.

[Card 90] Card, David N.; & Glass, Robert L. *Measuring Software Design Quality.* Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1990.

[Card 91] Card, David N. "What Makes a Software Measure Successful?" *American Programmer 4,* 9 (September 1991): 2-8.

[Decker 91] Decker, William J.; Baumert, John H.; Card, David N.; Wheeler, J. Ladd; & Wood, Richard J. *SEAS Software Measurement System Handbook* (CSC/TR-89/6166). Beltsville, MD: Computer Sciences Corporation, 1991.

[DeMarco 82] DeMarco, Tom. *Controlling Software Projects: Management, Measurement, and Estimation.* New York, NY: Yourdon Press, 1982.

[Deming 82] Deming, W. E. *Quality, Productivity, and Competitive Position.* Cambridge, MA: MIT Press, 1982.

[DoD 80] Departments of the Air Force, the Army, the Navy, and the Defense Logistic Agency. *Cost/Schedule Control Systems Criteria Joint Implementation Guide* (AFSC 173-5, AFLCP 173-5, DARCOM-P 715-5, NAVMAT P5240, DLAH 8315.2). October 1, 1980.

[DSDM 89] Computer Sciences Corporation. *Digital System Development Methodology, Version 3.* Falls Church, VA: December 1989.

[Dunn 90] Dunn, Robert H. *Software Quality: Concepts and Plans.* Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1990.

[Florac 92] Florac, William A.; The Quality Subgroup of the Software Metrics Definition Working Group; & the Software Process Measurement Project Team. *Software Quality Measurement: A Framework for Counting Problems and Defects* (CMU/SEI-92-TR-22, ESC-TR-22). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, September 1992.

[Gilb 77] Gilb, Tom. *Software Metrics.* Cambridge, MA: Winthrop Publishers, Inc., 1977.

[Goethert 92] Goethert, Wolfhart B.; Bailey, Elizabeth K.; Busby, Mary B.; The Effort and Schedule Subgroup of the Software Metrics Definition Working Group; & the Software Process Measurement Project Team. *Software Effort and Schedule Measurement: A Framework for Counting Staff-Hours and Reporting Schedule Information* (CMU/SEI-92-TR-21, ESC-TR-21). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, September 1992.

[Grady 87] Grady, Robert B.; & Caswell, Deborah L. *Software Metrics: Establishing a Company-Wide Program.* Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1987.

[Humphrey 89] Humphrey, Watts S. *Managing the Software Process.* Reading, MA: Addison-Wesley, 1989.

[IEEE 610] Institute of Electrical and Electronics Engineers. *IEEE Standard Glossary of Software Engineering Terminology* (ANSI/IEEE std 610.12-1990). New York, NY: December 10, 1990.

[IEEE 1044] Institute of Electrical and Electronics Engineers. *A Standard Classification for Software Errors, Faults, and Failures (Draft)* (IEEE P1044/D3). New York, NY: 1987.

[IEEE 1045] Institute of Electrical and Electronics Engineers. *Standard for Software Productivity Metrics (Draft)* (IEEE P1045/D4.0). New York, NY: December 1990.

[IEEE 1061] Institute of Electrical and Electronics Engineers. *Standard for Software Quality Metrics Methodology (Draft)* (IEEE P1061/D21). New York, NY: 1990.

[Jones 91] Jones, Capers. *Applied Software Measurement.* New York, NY: McGraw-Hill, Inc., 1991.

[Kuntzmann 92] Kuntzmann-Combelles, Annie; Comer, Peter; Holdsworth, Jacqueline; & Shirlaw, Stephen. *Metrics Users' Handbook.* Cambridge, England: Application of Metrics in Industry (AMI), 1992.

[Landis 90] Landis, Linda; McGarry, Frank; Waligora, Sharon; Pajerski, Rose; Stark, Mike; Kester, Rush; McDermott, Tim; & Miller, John. *Manager's Handbook for Software Development, Revision 1* (SEL-84-101). Greenbelt, MD: NASA Goddard Space Flight Center, 1990.

[Lockyer 84] Lockyer, Keith. *Critical Path Analysis and Other Project Network Techniques,* 4th ed. London, England: Pitman Publishing, 1984.

[London 89] London, Manuel. *Managing the Training Enterprise: High-Quality, Cost-Effective Employee Training in Organizations.* San Francisco, CA: Jossey-Bass, 1989.

[Mays 90] Mays, R.G.; Holloway, G.J. & Studinski, D.P. "Experiences with Defect Prevention." *IBM Systems Journal 29,* 1 (1990): 4-32.

[Murine 88] Murine, Gerald E. "Integrating Software Quality Metrics with Software QA." *Quality Progress 21,* 11 (November 1988): 38-43.

[Park 92] Park, Robert E.; The Size Subgroup of the Software Metrics Definition Working Group; & the Software Process Measurement Project Team. *Software Size Measurement: A Framework for Counting Source Statements* (CMU/SEI-92-TR-20, ESC-TR-20). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, September 1992.

[Paulk 91] Paulk, Mark C.; Curtis, Bill; Chrissis, Mary Beth; Averill, Edward L.; Bamberger, Judy; Kasse, Timothy C.; Konrad, Mike; Perdue, Jeffrey R.; Weber, Charles V.; & Withey James V. *Capability Maturity Model for Software* (CMU/SEI-91-TR-24, ADA240603). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1991.

[Pfleeger 89] Pfleeger, Shari Lawrence. *Recommendations for an Initial Set of Software Metrics* (CTC-TR-89-017). Chantilly, VA: Contel, 1989.

[Pyzdek 84] Pyzdek, Thomas. *An SPC Primer.* Tucson, AZ: Quality America, Inc., 1984.

[Pyzdek 89] Pyzdek, Thomas. *Pyzdek's Guide to SPC. Vol. 1, Fundamentals.* Tucson, AZ: Quality America, Inc., 1989.

[Rifkin 91] Rifkin, Stan; & Cox, Charles. *Measurement in Practice* (CMU/SEI-91-TR-16, ADA 241781). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1991.

[Rozum 92] Rozum, James A.; & Software Acquisition Metrics Working Group. *Software Measurement Concepts for Acquisition Program Managers* (CMU/SEI-92-TR-11, ESC-TR-11). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, September 1992.

[Schulmeyer 87] Schulmeyer, G. Gordon; & McManus, James I. *Handbook of Software Quality Assurance.* New York, NY: Van Nostrand Reinhold Company, Inc., 1987.

[Schultz 88] Schultz, Herman P. *Software Management Metrics* (ESD-TR-88-001). Bedford, MA: MITRE Corporation, 1988.

[STEP 91] Betz, Henry P.; & O'Neill, Patrick J. *Army Software Test and Evaluation Panel (STEP) Software Metrics Initiatives Report (Draft).* March 1991.

[Vincent 88] Vincent, James; Waters, Albert; & Sinclair, John. *Software Quality Assurance. Vol. 1, Practice and Implementation.* Englewood Cliffs, NJ: Prentice-Hall, Inc., 1988.

[Weber 91] Weber, Charles V.; Paulk, Mark C.; Wise, Cynthia J.; & Withey, James V. *Key Practices of the Capability Maturity Model* (CMU/SEI-TR-25, ADA240604). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1991.

Appendix A: Acronyms

ACWP	actual cost of work performed
BCAC	budgeted cost at completion
BCWP	budgeted cost for work performed
BCWS	budgeted cost for work scheduled
CDR	critical design review
CMM	Capability Maturity Model for Software
CPI	cost performance index
CSCI	computer software configuration item
CSU	computer software unit
CV	cost variance
IEEE	Institute of Electrical and Electronics Engineers, Inc.
I/O	input/output
Kbytes	thousand bytes
KLOC	thousand lines of code
KSLOC	thousand source lines of code
LOC	line of code
PCA	physical configuration audit
PERT	program evaluation and review technique
PDR	preliminary design review
RR	requirements review
SDR	system design review
SPC	statistical process control
SEI	Software Engineering Institute
SEPG	software engineering process group
SLOC	source line(s) of code
SPI	schedule performance index
SSR	software specification review
SV	schedule variance
TBD	to-be-determined

TRR	test readiness review
WBS	work breakdown structure

Appendix B: Definitions

action item - Any review discrepancy, clarification, or issue that must be resolved by the project or the customer. In the context of this document, an action item arises out of formal reviews with the customer and from peer reviews. The review result indicator only tracks action items from the formal reviews.

actual cost of work performed - The costs actually incurred and recorded in accomplishing the work performed within a given time period [DoD 80].

budgeted cost at completion - The sum of the budgeted cost for all work to be performed on the project.

budgeted cost for work performed - The sum of the budgets for completed work packages and completed portions of open work packages, plus the appropriate portion of the budgets for level of effort and apportioned effort [DoD 80].

budgeted cost for work scheduled - The sum of the budgets for all work packages, planning packages, etc., scheduled to be accomplished (including in-process work packages), plus the amount of effort and apportioned effort scheduled to be accomplished within a given time period [DoD 80].

class - The number of times a particular course is offered. For example, there may be six classes of the requirements analysis course in one month.

cost account - A management control point at which actual costs can be accumulated and compared to budgeted cost for work performed. A cost account is a natural control point for cost/schedule planning and control since it represents the work assigned to one responsible organizational element on one contract work breakdown structure element [DoD 80].

cost performance index - The ratio of the budgeted cost for work performed and the actual cost of the work performed, expressed as a percentage.

cost variance - The difference between the budgeted cost for work performed and the actual cost of work performed.

course - An offering in particular subject.

data - Something given or admitted, especially as a basis for reasoning or inference; factual material used as a basis especially for discussion or decision; something used as a basis for calculating or measuring.

defect - A product's inconsistency with its specification. Examples include such things as omissions and imperfections found in software during the early life-cycle phases and faults contained in software sufficiently mature for test or operation.

error - (1) The difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. (2) An

incorrect step, process, or data definition. (3) An incorrect result. (4) A human action that produces an incorrect result [IEEE 610].

estimated cost at completion - Actual costs to date plus the estimate of costs for authorized work remaining (based on [DoD 80]).

failure - The inability of a system or component to perform its required functions within specified performance requirements [IEEE 610].

fault - An incorrect step, process, or data definition in a computer program. In common usage, the terms "error" and "bug" are used to express this meaning [IEEE 610].

first-line software manager - A manager who has direct management responsibility (including providing technical direction and administering the personnel and salary functions) for the staff and activities of a single department of software engineers and other related staff.

formal review - A formal meeting at which a product is presented to the end user, customer, or other interested parties for comment and approval. It can also be a review of the management and technical activities and progress of the hardware/software development project.

indicator - A representation of measurement data that provides insight into software development processes and/or software process improvement activities.

labor category - A way of classifying labor. Labor categories can be direct or indirect, or broken down by job classification or skill.

measure - *n.* A standard or unit of measurement; the extent, dimensions, capacity, etc. of anything, especially as determined by a standard; an act or process of measuring; a result of measurement. *v.* To ascertain the quantity, mass, extent, or degree of something in terms of a standard unit or fixed amount, usually by means of an instrument or process; to compute the size of something from dimensional measurements; to estimate the extent, strength, worth, or character of something; to take measurements.

measurement - The act or process of measuring something. Also a result, such as a figure expressing the extent or value that is obtained by measuring.

metric - In this document, metric is used as a synonym for measure.

mid-level software manager - A manager who reports directly or indirectly (i.e., through another manager) to the project software manager and who has direct management responsibility (including providing technical and management direction and administering the personnel and salary functions) for other software managers.

noncompliance - The failure of an intermediate or final product to comply with a standard or of an activity to comply with its process.

priority - The level of importance assigned to an item by an individual. Priority is related to how quickly an item gets addressed.

problem - An unsettled question arising from a situation where it appears that a change to the software, its documentation, or related hardware is necessary for successful test or operation of the system [IEEE P1044].

process - A series of actions, changes, or functions that achieve an end or result.

project software manager - A manager who has total responsibility for all the software activities for the entire project. The project software manager is the person the project manager deals with in terms of software commitments, etc., and the person who controls all the software development staff for a project.

quality audit - The review of a product or process for compliance with standards, procedures, and progress reporting policy by an independent software quality assurance group.

schedule performance index - The ratio of the budgeted cost for the work performed and the budgeted cost for the work scheduled, expressed as a percentage.

schedule variance - The difference between the budgeted cost for the work performed and the budgeted cost for work scheduled.

senior manager - A manager at a high enough level that his/her primary focus would be expected to be the long-term vitality of the company and organization, rather than short-term project and contractual concerns and pressures. In general, a senior manager for engineering would have responsibility for multiple projects.

severity - The degree of impact that a requirement, module, error, fault, failure, or other item has on the development or operation of a system [IEEE 610].

software engineering process group - A group of specialists who facilitate the definition and improvement of the software process used by the organization.

staff-hour - An hour of time expended by a member of the staff [IEEE 1045].

statistical process control - The use of statistical methods to identify the existence of special causes of variation in a process.

trouble report - A document (electronic or hard copy) used to recognize, record, track, and close anomalies detected in the software and its accompanying documentation. In this document, a trouble report is restricted to that written during integration and test, and the acceptance test activities.

turnover - The number of unplanned staff losses during a reporting period.

variance - The difference between planned and actual performance which requires further review, analysis, or action [DoD 80].

work breakdown structure - A product-oriented family tree division of hardware, software, services, and other work tasks which organizes, defines, and graphically displays the product to be produced as well as the work to be accomplished to achieve the specified product [DoD 80].

work package - Detailed short-span jobs, or material items, identified by the contractor for accomplishing work required to complete the contract [DoD 80].

Appendix C: Mapping of Software Measurement in the Key Practices of the Capability Maturity Model to Indicator Categories

This appendix provides the software measurement references contained within the key process areas of the Capability Maturity Model for Software (CMM) [Weber 91]. The information is provided in five columns. The first column repeats the goals of the key process area from the CMM. The second column list questions that can be asked about the goals. In most cases, these questions are concerned with measurement. The third column lists the measures that are explicitly stated or inferred from the CMM text. The reference for each measure is given in the fourth column. Each entry consists of a letter indicating whether the measure is explicitly stated (E) or inferred (I) in the CMM text and a citation code. This code has the format

key process area.common feature.top-level key practice number.subordinate key practice number.item in subordinate key practice statement

Thus, E RM.MO.1.2.1 means that the measure *number of requirements changes over time* is explicitly stated (E) in the Requirements Management (RM) key process area, the Monitoring implementation (MO) common feature, top-level key practice statement 1, subordinate key practice statement 2, and item 1 within the subordinate key practice statement. The code key for the CMM reference is given on the next page.

The five common features of the key practices for each key process area are the following:

- *Commitment to perform* specifies the actions the organization must take to ensure that the process is established and will endure.
- *Ability to perform* specifies the preconditions that must exist in the project or organization to implement the process competently.
- *Activities performed* specifies the steps that must be performed to effectively establish the key process area.
- *Monitoring implementation* specifies the steps that must be performed to measure the process, analyze the measurements, and take action based on the results.
- *Verifying implementation* specifies the steps that must be performed to guide and ensure that the activities are performed in compliance with the process that has been specified.

The last column lists the indicator category to which the measurement applies.

CMM Reference Key

Abbreviation	Meaning
Key Process Area:	
RM	Requirements Management
PP	Software Project Planning
TO	Software Tracking and Oversight
SM	Software Subcontract Management
QA	Software Quality Assurance
CM	Software Configuration Management
PF	Organization Process Focus
PD	Organization Process Definition
TR	Training
IM	Integrated Software Management
PE	Software Product Engineering
IC	Intergroup Coordination
PR	Peer Reviews
PA	Process Measurement and Analysis
QM	Quality Management
DP	Defect Prevention
TI	Technology Innovation
PC	Process Change Management
Common Feature:	
CO	Commitment to perform
AB	Ability to perform
AC	Activities performed
MO	Monitoring implementation
VE	Verifying implementation

Requirements Management

Key Process Area Goals	Questions
The system requirements allocated to software provide a clearly stated, verifiable, and testable foundation for software engineering and software management.	Does the requirements management process track the number of changes to the requirements?
The allocated requirements define the scope of the software effort.	Does the requirements management process monitor the completeness of the requirements?
The allocated requirements and changes are incorporated into the software plans, products, and activities in an orderly manner.	Does the software development plan appropriately address requirements implementation?

Requirements Management

Measurement	CMM Category	Indicator Category
Total number of allocated requirements, over time	I RM.AC.1	Requirements stability
Number of allocated requirements, over time, by type:	I RM.AC.1	Requirements stability
Functional		
Performance		
Operational		
Programmatic		
Cost for managing requirements	E RM.MO.1	Cost
Schedule of activities for managing requirements	E RM.MO.1	Progress
Total number of changes, over time, from each major source:	E RM.MO.1.2.1	Requirements stability
Customer		
End user		
Software engineering group		
System engineering group		
System test group		
Total number of changes, over time	E RM.MO.1.2.2	Requirements stability
Proposed		
Open		
Approved		
Incorporated into baseline		
Number of items affected by requirements change	I RM.AC.4.5	Requirements stability
Number of SQA reviews	E RM.VE.3	SQA audit results
Number of SQA audits	E RM.VE.3	SQA audit results
Number of deviations and product deficiencies	I RM.VE.3	SQA audit results
Open		
Closed		
Type		

Software Project Planning

Key Process Area Goals	Questions
A plan is developed that appropriately and realistically covers the software activities and commitments.	Has the software planning process resulted in a software development plan that has the planned budget, effort, and schedule consistent with each other and the software requirements?
All affected groups and individuals understand the software estimates and plans and commit to support them.	Are the software subcontractor's planned budget, effort, and schedule consistent with each other, the software requirements, and the software development plan?
The software estimates and plans are documented for use in tracking the software activities and commitments.	

Software Project Planning

Measurement	CMM Category	Indicator Category
Identification of software products to be developed, including:	E PP.AC.7.4	Progress
Major products used by software engineering group		
Products used by other groups		
Products for external delivery		
Size estimates for software products and activities	E PP.AC.7.5, E PP.AC.9.1	Size stability
Staff resource estimates by	E PP.AC.7.6, E PP.AC.10.3.3	Effort
Life-cycle stage		
Task		
Spread over time		
Skill level	E PP.AC.10.3.4	
Effort for project planning activities	E PP.MO.1	Effort
Cost estimates by	E PP.AC.10.3.3	Cost
Life-cycle stage		
Task		
Spread over time		
Cost for project planning activities	E PP.MO.1	Cost
Software project schedules	E PP.AC.7.7, E PP.AC.12.3, E PP.AC.12.4, E PP.MO.1	Progress
Total number of risks	E PP.AC.7.8	
Number of risks by type	I PP.AC.7.8, E PP.AC.13	
Technical		
Cost		
Resource		
Schedule		
Critical computer resource estimates	E PP.AC.11.1, E PP.AC.11.2	Computer resource utilization
Memory capacity		
Computer process use		
Communication channel capacity		
Number of SQA reviews	E PP.VE.2	SQA audit results
Number of SQA audits	E PP.VE.2	SQA audit results
Number of deviations and product deficiencies	I PP.VE.2	SQA audit results
Open		
Closed		
Type		
Number of staff trained	E PP.AB.5	Training

Software Project Tracking and Oversight

Key Process Area Goals	Questions
Actual results and performance of the software project are tracked against documented and approved plans.	Are the actual results and performance of the software project tracked against the software development plan?
Corrective actions are taken when the actual results and performance of the software project deviate significantly from the plans.	Are the planned budget, effort, and schedule revised when necessary with the procedure used to derive the original budget, effort, and schedule?
Changes to software commitments are understood and agreed to by all affected groups and individuals.	Is there a mechanism for the review of the cost, schedule, resource, and technical risk status of the software development project?

Software Project Tracking and Oversight

Measurement	CMM Category	Indicator Category
Size for software products and activities	E TO.AC.4.1	Progress
Actual size of	E TO.AC.4.2	Progress
Generated code		
Fully tested code		
Delivered code		
Units of delivered documentation	E TO.AC.4.3	Progress, Size stability
Total software size	E TO.AC.4.4	Progress, Size stability
Actual costs over time	E TO.AC.5.1, E TO.MO.1.3, E TO.VE.1.2, E TO.VE.2.1	Cost
Software costs per element	E TO.AC.5.2, E TO.MO.1.2	Cost
Effort and staffing	E TO.AC.5.3, E TO.MO.1.3, E TO.VE.1.2, E TO.VE.2.1	Effort
Critical computer resource	E TO.AC.6.1, E TO.VE.1.3	Computer resource utilization
Memory capacity		
Computer process use		
Communication channel capacity		
Schedule	E TO.AC.7, E TO.MO.1.1, E TO.VE.1.2, E TO.VE.2.1	Progress
Software units designed	E TO.AC.7.2	Progress
Software units coded	E TO.AC.7.2	Progress
Software units tested	E TO.AC.7.2	Progress
Software units integrated	E TO.AC.7.2	Progress
Completion dates of test executions	E TO.AC.7.3	Progress
Software activity completion dates	E TO.AC.7.4	Progress
System release contents	E TO.AC.8.2	Progress, Requirements stability, Size stability
Trouble reports	E TO.AC.8.3	Trouble reports
Software risks	E TO.AC.9, E TO.VE.1.6, E TO.VE.2.3	
Technical		
Cost		
Resource		
Schedule		
Number of action items	E TO.AC.12.6	Review results

Software Project Tracking and Oversight (Continued)

Measurement	CMM Category	Indicator Category
Number of SQA reviews	E TO.VE.3	SQA audit results
Number of SQA audits	E TO.VE.3	SQA audit results
Number of deviations and product deficiencies	I TO.VE.3	SQA audit results
Open		
Closed		
Type		
Number of staff trained	E TO.AB.6, E TO.AB.7	Training

Software Subcontract Management

Key Process Area Goals	Questions
The prime contractor selects qualified subcontractors.	Are the actual results and performance of the software subcontractor tracked against the software development plan?
The software standards, procedures, and product requirements for the subcontractor comply with the prime contractor's commitments.	Is there a mechanism for the review of the status of the cost, effort, schedule, and technical risks of the software subcontractor?
Commitments between the prime contractor and subcontractor are understood and agreed to by both parties.	
The prime contractor tracks the subcontractor's actual results and performance against the commitments.	

Software Subcontract Management

Measurement	CMM Category	Indicator Category
Size estimates	E SM.AC.3.7.1	Progress, Size stability
Cost estimates	E SM.AC.3.7.2	Progress
Schedule estimates	E SM.AC.3.7.3	Progress
Critical computer resource estimates	E SM.AC.3.7.4	Computer resource utilization
Actual subcontractor costs	E SM.AC.7.2	Cost
Actual subcontractor effort	E SM.AC.7.2	Effort
Actual subcontractor schedule performance	E SM.AC.7.2	Progress
Subcontractor critical computer resources	E SM.AC.7.3	Computer resource utilization
Action items	E SM.AC.9.3	Review results
Software risks	E SM.AC.9.4	
Number of SQA reviews	E SM.AC.10.2, E SM.VE.3	SQA audit results
Number of SQA audits	E SM.AC.11.3, E SM.VE.3	SQA audit results
Number of deviations and product deficiencies	I SM.AC.10.2, I SM.AC.11.3, I SM.VE.3	SQA audit results
Open		
Closed		
Type		
Cost for managing subcontract	E SM.MO.1	Cost
Schedule status of managing subcontract activities	E SM.MO.1	Progress
Number of staff trained	E SM.AB.2, E SM.AB.3	Training

Software Quality Assurance

Key Process Area Goals	Questions
Compliance of the software product and software process with applicable standards, procedures, and product requirements is independently confirmed.	Are independent audits conducted for each step of the software development process?
When there are compliance problems, management is aware of them.	Are standards and procedures applied on the software development project?
Senior management addresses noncompliance issues.	Are independent audits conducted for the software subcontractor to ensure compliance with the software development plan?

Software Quality Assurance

Measurement	CMM Category	Indicator Category
Number of SQA reviews	E QA.AC.2.5, E QA.AC.4, E QA.MO.1.3	SQA audit results
Number of SQA audits	E QA.AC.2.5, E QA.AC.4, E QA.MO.1.3	SQA audit results
Number of deviations and product deficiencies	E QA.AC.4.2, E QA.AC.5.3	SQA audit results
Open		
Closed		
Type		
Cost of SQA activities	E QA.MO.1.2	Cost
Schedule status for SQA activities	E QA.MO.1.1, E QA.MO.1.2	Progress
Effort expended for SQA activities	E QA.MO.1.2	Effort
Number of staff trained	E QA.AB.2, E QA.AB.3	Training

Software Configuration Management

Key Process Area Goals	Questions
Controlled and stable baselines are established for planning, managing, and building the system.	Is there a mechanism for identifying software baseline(s)?
The integrity of the system's configuration is controlled over time.	Does the configuration management process track the number of changes to the software baseline(s)?
The status and content of the software baselines are known.	

Software Configuration Management

Measurement	CMM Category	Indicator Category
Total number of configuration items	E CM.AC.5	Progress, Requirements stability, Size stability
Number of configuration items by type	E CM.AC.5.1	Progress, Requirements stability, Size stability
Number of change requests, over time	E CM.AC.6	Requirements stability
Number of trouble reports	E CM.AC.6	Trouble reports
Open		
Approved		
Closed		
Cost for SCM activities	E CM.MO.1.2	Cost
Effort for SCM activities	E CM.MO.1.2	Effort
Schedule status for SCM activities	E CM.MO.1.1, E CM.MO.1.2	Progress
Number of baseline audits	E CM.AC.11, E CM.VE.4	SQA audit results
Number of reviews	E CM.CO.1.5, E CM.VE.3, E CM.AC.7.3, E CM.VE.4	SQA audit results
Number of SQA audits	E CM.VE.4	SQA audit results
Number of deviations and product deficiencies	I CM.AC.7.3, I CM.VE.3, I CM.VE.4, I CM.AC.11	SQA audit results
Open		
Closed		
Type		
Number of staff trained	E CM.AB.4, E CM.AB.5	Training

Organization Process Focus

Key Process Area Goals	Questions
Current strengths and weaknesses of the organization's software process are understood and plans are established to systematically address the weaknesses.	Has a software engineering process group (SEPG) been established?
	Has the SEPG a charter that directs it to focus on the organization's process?
A group is established with appropriate knowledge, skills, and resources to define a standard software process for the organization.	
The organization provides the resources and support needed to record and analyze the use of the organization's standard software process to maintain and improve it.	

Organization Process Focus

Measurement	CMM Category	Indicator Category
Schedule for process definition and improvement activities	E PF.MO.1.1	Progress
Effort expended for process definition and improvement activities	E PF.MO.1.1	Effort
Cost for process definition and improvement activities	E PF.MO.1.1	Cost
Number of staff trained	E PF.AB.2, E PF.AB.3	Training

Organization Process Definition

Key Process Area Goals	Questions
A standard software process for the organization is defined and maintained as a basis for stabilizing, analyzing, and improving the performance of the software projects.	Is the standard software process stable?
	Are process problems the result of poor process definition or of improper implementation of the process?
Specifications of common software processes and documented process experiences from past and current projects are collected and available.	

Organization Process Definition

Measurement	CMM Category	Indicator Category
Number of changes to the software software process	E PD.AC.7.2	Process stability
Number of waivers to the software process	E PD.AC.9.2	Process stability
Schedule for process definition activities	E PD.MO.1.1	Progress
Effort expended for process definition activities	I PD.MO.1	Effort
Cost for process definition activities	E PD.MO.1.2	Cost
Number of SQA reviews	E PD.VE.1	SQA audit results
Number of SQA audits	E PD.VE.1	SQA audit results
Number of deviations and product deficiencies	I PD.VE.1	SQA audit results
Open		
Closed		
Type		
Number of staff trained	E PD.AB.2, E PD.AB.3, E PD.AB.4	Training

Training Program

Key Process Area Goals	Questions
The staff and managers have the skills and knowledge to perform their jobs.	Does the staff have the appropriate skill mix for the project?
	How many of the staff are being trained?
The staff and managers effectively use, or are prepared to use, the capabilities and features of the existing and planned work environment.	How many of the staff have completed their training programs?
	How is the project performing with respect to its training plan?
The staff and managers are provided with opportunities to improve their professional skills.	Who is attending training?
	Are the appropriate people being trained?
	What is the quality/effectiveness of the training?
	How many waivers from training have been requested?

Training Program

Measurement	CMM Category	Indicator Category
Number of waivers	E TR.AC.4, E TR.MO.1.3	Training
Approved over time for each project		
Approved over time for each course		
Number of staff trained	E TR.AC.8.1, E TR.MO.1.1, E TR.AB.2, E TR.AB.3	Training
Number who completed their designated required courses	E TR.AC.8.1, E TR.MO.1.1, E TR.AB.2, E TR.AB.3	Training
Number of courses offered	E TR.MO.1.2	Training, Progress
Course quality	E TR.MO.2	Training
Number of SQA reviews	E TR.VE.4	SQA audit results
Number of SQA audits	E TR.VE.4	SQA audit results
Number of deviations and product deficiencies	I TR.VE.4	SQA audit results
Open		
Closed		
Type		

Integrated Software Management

Key Process Area Goals	Questions
The planning and managing of each software project is based on the organization's standard software process.	Has the software planning process resulted in a software development plan that has the planned budget, effort, and schedule consistent with each other and the software requirements?
Technical and management data from past and current projects are available and used to effectively and efficiently estimate, plan, track, and replan the software projects.	Are the software subcontractor's planned budget, effort, and schedule consistent with each other, the software requirements, and the software development plan?
	Are the actual results and performance of the software project tracked against the software development plan?
	Are the planned budget, effort, and schedule revised when necessary with the procedure used to derive the original budget, effort, and schedule?
	Is there a mechanism for the review of the cost, schedule, resource, and technical risk status of the software development project?
	Does the requirements management process track the number of changes to the requirements?
	Does the requirements management process monitor the completeness of the requirements?
	Does the software development plan appropriately address requirements implementation?

Integrated Software Management

Measurement	CMM Category	Indicator Category
Total number of allocated requirements, over time	E IM.AB.2	Requirements stability
Number of waivers to the software process	E IM.AC.1.2.2	Process stability
Approved		
Rejected		
Number of waivers to the contract software process requirements	E IM.AC.1.3	Process stability
Approved		
Rejected		
Number of software process change requests	E IM.AC.2.2	Process stability
Open		
Approved		
Rejected		
Incorporated		
Number of action items	I IM.AC.3	Review results
Open		
Closed		
Type		
Priority		
Software size	E IM.AC.6.1, E IM.AC.6.6.1, E IM.MO.1.1	Size stability, Progress
New		
Off-the-shelf		
Reused		
Project software costs	E IM.AC.7, E IM.MO.1.3	Cost
Critical computer resources	E IM.AC.8, E IM.MO.3.3	Computer resource utilization
Memory capacity		
Computer process use		
Communication channel capacity		
Software project schedule	E IM.AC.9, E IM.MO.1.2, E IM.MO.3.2	Progress
Total number of risks	E IM.AC.10, E IM.MO.3, E PP.AC.7.8	
Number of risks by type	I IM.AC.10, I PP.AC.7.8, E PP.AC.13, E TO.AC.9, E TO.VE.1.6, E TO.VE.2.3, E SM.AC.9.4	

Integrated Software Management (Continued)

Key Process Area Goals	Questions
	Are independent audits conducted for each step of the software development process?
	Are standards and procedures applied on the software development project?
	Is there a mechanism for identifying software baseline(s)?
	Does the configuration management process track the number of changes to the software baseline(s)?

Integrated Software Management (Continued)

Measurement	CMM Category	Indicator Category
Technical		
Cost		
Resource		
Schedule		
Number of defects	E IM.MO.1.4	Peer review results
Number of replans	E IM.MO.2	
Cause		
Magnitude		
Staff resources by	E IM.MO.3.1	Effort
Life-cycle stage		
Task		
Skill level		
Number of SQA reviews	E IM.VE.3	SQA audit results
Number of SQA audits	E IM.VE.3	SQA audit results
Number of deviations and product deficiencies	I IM.VE.3	SQA audit results
Open		
Closed		
Type		
Number of staff trained	E IM.AB.5	Training

Software Product Engineering

Key Process Area Goals	Questions
Software engineering issues for the product and the process are properly addressed in the system requirements and system design.	Does the quality of the product indicate that the product is ready to proceed to the next life-cycle stage or that it is ready for release to the customer?
The software engineering activities are well-defined, integrated, and used consistently to produce software systems.	Will undetected or unresolved problems in the product lead to more problems in the next life-cycle stage?
State-of-the-practice software engineering tools and methods are used, as appropriate, to build and maintain the software system.	Does the number of trouble reports indicate that the software product should be reworked before proceeding to the next life-cycle stage?
Software engineering products that are consistent with each other and appropriate for building and maintaining the software system are systematically developed.	Do the types of defects suggest areas for process changes?
	Is the testing activity complete?
	Are project personnel addressing trouble reports, requirements change requests, and waiver requests in a timely manner?
	Are the numbers of requirements changes, to-be-determineds, trouble reports, and waiver requests decreasing with time?
	What category of requirements are responsible for the majority of the requirements changes?

Software Product Engineering

Measurement	CMM Category	Indicator Category
Total number of allocated requirements, over time	E PE.AC.3.5, E PE.MO.1.2, E PE.MO.2.1	Requirements stability
Number of allocated requirements, over time, by type:	E PE.AC.3.1.1, E PE.MO.1.2, E PE.MO.2.1	Requirements stability
Functional		
Performance		
Operational		
Programmatic		
Total number of changes, over time, from each major source:	E PE.MO.2.3	Requirements stability
Customer		
End user		
Software engineering group		
System engineering group		
System test group		
Total number of changes, over time	E PE.MO.2.3	Requirements stability, Effort, Cost
Proposed		
Open		
Approved		
Incorporated into baseline		
Trouble reports	E PE.MO.2.2, E PE.VE.3.9	Trouble reports
Open		
Approved		
Closed		
Total number of defects detected	E PE.AC.10, E PE.MO.1.1	Trouble reports, Peer review results
Number of defects		Trouble reports, Peer review results
by type/category	E PE.AC.10.2, E PE.MO.1.1	
by severity	E PE.AC.10.3, E PE.MO.1.1	
by life-cycle stage	E PE.MO.1.1	
by activity introduced	E PE.AC.10.6	
Number of units affected by defect	E PE.AC.10.5	Trouble reports, Peer review results
Number of units containing defect	E PE.AC.10.4	Trouble reports, Peer review results
Number of changes to the software process	E PE.AC.11.4	Process stability
Number of SQA reviews	E PE.VE.3	SQA audit results
Number of SQA audits	E PE.VE.3	SQA audit results

Software Product Engineering (Continued)

Measurement	CMM Category	Indicator Category
Number of deviations and product deficiencies	I PE.VE.3	SQA audit results
Open		
Closed		
Type		
Number of staff trained	E PE.AB.2, E PE.AB.3	Training

Intergroup Coordination

Key Process Area Goals	Questions
The project's technical goals and objectives are understood and agreed to by its staff and managers.	Are the project personnel working together effectively to accomplish the software development plan?
The responsibilities assigned to each of the project groups and the working interfaces between these groups are known to all groups.	Are the project personnel following the standard software process?
The project groups are appropriately involved in intergroup activities and in identifying, tracking, and addressing intergroup issues.	
The project works as a team.	

Intergroup Coordination

Measurement	CMM Category	Indicator Category
Critical dependencies	E IG.AC.4	Progress
Cost of intergroup coordination activities	E IG.MO.1.1, E IG.MO.1.2	Cost
Effort of intergroup coordination activities	E IG.MO.1.1, E IG.MO.1.2	Effort
Schedule of intergroup coordination activities	E IG.MO.1.3, E IG.MO.1.4	Progress
Number of SQA reviews	E IG.VE.3	SQA audit results
Number of SQA audits	E IG.VE.3	SQA audit results
Number of deviations and product deficiencies	I IG.VE.3	SQA audit results
Open		
Closed		
Type		
Number of staff trained	E IG.AB.3, E IG.AB.4, E IG.AB.5	Training

Peer Reviews

Key Process Area Goals	Questions
Product defects are identified and fixed early in the life cycle.	Is there a peer review process in place that allows the detection of product defects early in the life cycle?
Appropriate product improvements are identified and implemented early in the life cycle.	Are action items resulting from peer reviews tracked to closure?
The staff members become more effective through a better understanding of their work products and knowledge of errors that can be prevented.	Are the defect data used to identify weaknesses in the software development process?
A rigorous group process for reviewing and evaluating product quality is established and used.	

Peer Reviews

Measurement	CMM Category	Indicator Category
Number of peer reviews	I PR.AC.1	Peer review results
Number of re-reviews	I PR.AC.3, E PR.MO.1	Peer review results
Action items	E PR.AC.2.5, E PR.AC.3.6	Peer review results
Open		
Closed		
Number of items reviewed at meeting	I PR.AC.3.1	Peer review results
Product size	E PR.AC.3.2, E PR.MO.1.3	Peer review results
Preparation lead time	E PR.MO.1.1	Peer review results
Preparation time for each reviewer	E PR.AC.3.3, E PR.MO.1.2	Peer review results
Length of time of review	E PR.AC.3.4, E PR.MO.1.6	Peer review results
Size of review team	I PR.AC.1.4, E PR.MO.1.4	Peer review results
Experience of review team	E PR.AC.1.4, E PR.MO.1.5	Peer review results
Structure of review team	E PR.AC.1.4	Peer review results
Number of defects detected	E PR.AC.3.5	Peer review results
Number of defects identified by type	E PR.AC.3.5	Peer review results
Rework effort	E PR.AC.3.7, E PR.MO.1.8	Peer review results
Number of SQA reviews	E PR.VE.1	SQA audit results
Number of SQA audits	E PR.VE.1	SQA audit results
Number of deviations and product deficiencies	I PR.VE.1	SQA audit results
Open		
Closed		
Type		
Number of staff trained	E PR.AB.1, E PR.AB.2	Training

Process Measurement and Analysis

Key Process Area Goals	Questions
The organization's standard software process is stable and under statistical process control.	Is the organization's standard software process stable and under statistical process control?
The relationship between product quality, productivity, and product development cycle time is understood in quantitative terms.	Are process variations analyzed and corrective action taken when necessary?
Special causes of process variation (i.e., variations attributable to specific applications of the process and not inherent in the process) are identified and controlled.	Are process and product metrics collected, analyzed, and used to monitor and control products and processes?

Process Measurement and Analysis

Measurement	CMM Category	Indicator Category
Number of staff trained	E PA .AB.3, E PA.AB.4	Training
Product and process metrics	E PA.AC.3.5	Progress, Effort, Cost, Quality, Stability, Computer resource results
Number of change requests	E PA.AC.7.1	Process stability
Cost of process measurement and analysis activities	E PA.MO.1	Cost
Schedule for process measurement and analysis activities	E PA.MO.1	Progress
Number of SQA reviews	E PA.VE.5	SQA audit results
Number of SQA audits	E PA.VE.5	SQA audit results
Number of deviations and product deficiencies	I PA.VE.5	SQA audit results
Open		
Closed		
Type		

Quality Management

Key Process Area Goals	Questions
Measurable goals and priorities for product quality are established and maintained for each software project through interaction with the customer, end users, and project groups.	Are quantitative product and process quality goals established and revised throughout the software life cycle?
Measurable goals for process quality are established for all groups involved in the software process.	Are quality metrics collected, analyzed, and used to monitor and control products and processes?
The software plans, design, and process are adjusted to bring forecasted process and product quality in line with the goals.	Are variations from the quality goals analyzed and corrective action taken when necessary?
Process measurements are used to manage the software project quantitatively.	

Quality Management

Measurement	CMM Category	Indicator Category
Number of trained staff	E QM.AB.2, E QM.AB.3	Training
Cost of achieving quality goals	E QM.AC.11.1, E QM.MO.1.2	Cost
Cost of quality	E QM.MO.1.1	Cost
Number of SQA reviews	E QM.VE.4	SQA audit results
Number of SQA audits	E QM.VE.4	SQA audit results
Number of deviations and product deficiencies	I QM.VE.4	SQA audit results
Open		
Closed		
Type		

Defect Prevention

Key Process Area Goals	Questions
Sources of product defects that are inherent or repeatedly occur in the software process activities are identified and eliminated.	What types of defects are being inserted? Which types of defects are being inserted the most? What is the effect on the defect insertion rate when defect prevention procedures are enacted?

Defect Prevention

Measurement	CMM Category	Indicator Category
Number of trained staff	E DP.AB.4	Training
Total number of defects detected	E DP.AC.2.2	Trouble reports, Peer review results, Defect prevention
Number of defects		
by type/category	E DP.AC.2.3, E DP.VE.1.1	Trouble reports, Peer review results, Defect prevention
by life-cycle stage	E DP AC.7.1, E DP.MO.2	Trouble reports, Peer review results, Defect prevention
by activity introduced	E DP AC.7.1, E DP.MO.2	Trouble reports, Peer review results, Defect prevention
Number of changes to the process	E DP.AC.6	Process stability
Number of changes to the software	E DP.AC.6	Size stability
Cost of defect prevention activities	E DP.MO.1.1, E DP MO.1.2, E DP.VE.1.5, E DP.VE.1.6	Cost
Schedule of defect prevention activities	E DP.MO.1.1	Progress
Effort spent on defect prevention activities	E DP.MO.1.2	Effort
Number of SQA reviews	E DP.VE.2	SQA audit results
Number of SQA audits	E DP.VE.2	SQA audit results
Number of deviations and product deficiencies	I DP.VE.2	SQA audit results
Open		
Closed		
Type		

Technology Innovation

Key Process Area Goals	Questions
The organization has a software process and technology capability to allow it to develop or capitalize on the best available technologies in the industry.	Is the organization actively seeking and installing new technology in an orderly fashion?
Selection and transfer of new technology into the organization is orderly and thorough.	
Technology innovations are tied to quality and productivity improvements of the organization's standard software process.	

Technology Innovation

Measurement	CMM Category	Indicator Category
Product and process metrics	E TI.AB.3	Progress, Effort, Cost, Quality, Stability
Number of trained staff	E TI.AB.4, E TI.AC.1.5, E TI.AC.7.6	Training
Number of changes to the software process	E TI.AC.7	Process stability
Cost of technology innovation activities	E TI.MO.1, E TI.MO.2	Cost
Schedule of technology innovation activities	E TI.MO.2	Progress
Number of SQA reviews	E TI.VE.2	SQA audit results
Number of SQA audits	E TI.VE.2	SQA audit results
Number of deviations and product deficiencies	I TI.VE.2	SQA audit results
Open		
Closed		
Type		

Process Change Management

Key Process Area Goals	Questions
The organization's staff and managers are actively involved in setting quantitative, measurable improvement goals and in improving the software process.	Is the organization continually working to improve its standard software process?
	Is the organization meeting its software process improvement goals?
The organization's standard software process and the projects' defined software processes continually improve.	
The organization's staff and managers are able to use the evolving software processes and their supporting tools and methods properly and effectively.	

Process Change Management

Measurement	CMM Category	Indicator Category
Number of people trained	E PC.AB.4, E PC.AC.5, E PC.AC.9.7	Training
Cost of process improvement activities	E PC.AC.7	Cost
Effort of process improvement activities	E PC.AC.7	Effort
Number of process changes	E PC.AC.9.2.1 E PC.AC.9.2.5	Process stability, Progress
Number of software process change requests	E PC.AC.1.6, E PC.AC.1.7, E PC.AC.5, E PC.AC.6,	Process stability
Open	E PC.AC.10, E PC.MO.1.1	
Approved		
Rejected		
Incorporated		
Number of SQA reviews	E PC.VE.2	SQA audit results
Number of SQA audits	E PC.VE.2	SQA audit results
Number of deviations and product deficiencies	I PC.VE.2	SQA audit results
Open		
Closed		
Type		

Appendix D: Indicator Categories Traced to Capability Maturity Model Key Practices

This appendix traces the software indicators to the software measurement references contained within the key process areas of the Capability Maturity Model (CMM) [Weber 91]. The information is in two columns. The first column gives the software indicator while the second column lists the CMM reference. Each entry consists of a letter indicating whether the measure is explicitly stated (E) or inferred (I) in the CMM text and a citation code. This code has the format

key process area.common feature.top-level key practice number.subordinate key practice number.item in subordinate key practice statement

Thus, E RM.MO.1.2.1 means that the measure *number of requirements changes over time* is explicitly stated (E) in the Requirements Management (RM) key process area, the Monitoring implementation (MO) common feature, top-level key practice statement 1, subordinate key practice statement 2, and item 1 within the subordinate key practice statement. The code key for the CMM reference is given on the next page.

The five common features of the key practices for each key process area are the following:

- *Commitment to perform* specifies the actions the organization must take to ensure that the process is established and will endure.
- *Ability to perform* specifies the preconditions that must exist in the project or organization to implement the process competently.
- *Activities performed* specifies the steps that must be performed to effectively establish the key process area.
- *Monitoring implementation* specifies the steps that must be performed to measure the process, analyze the measurements, and take action based on the results.
- *Verifying implementation* specifies the steps that must be performed to guide and ensure that the activities are performed in compliance with the process that has been specified.

The table ends with a risk indicator category, but as noted in the text, no indicator is given in this document for risk. The reference to risk is included for completeness.

CMM Reference Key

Abbreviation		Meaning
Key Process Area:		
	RM	Requirements Management
	PP	Software Project Planning
	TO	Software Tracking and Oversight
	SM	Software Subcontract Management
	QA	Software Quality Assurance
	CM	Software Configuration Management
	PF	Organization Process Focus
	PD	Organization Process Definition
	TR	Training
	IM	Integrated Software Management
	PE	Software Product Engineering
	IC	Intergroup Coordination
	PR	Peer Reviews
	PA	Process Measurement and Analysis
	QM	Quality Management
	DP	Defect Prevention
	TI	Technology Innovation
	PC	Process Change Management
Common Feature:		
	CO	Commitment to perform
	AB	Ability to perform
	AC	Activities performed
	MO	Monitoring implementation
	VE	Verifying implementation

Progress

Indicator	CMM Category
Schedule estimates	E SM.AC.3.7.3
Software project schedules	E PP.AC.7.7, E PP.AC.12.3, E PP.AC.12.4, E PP.MO.1, E TO.AC.7, E TO.MO.1.1, E TO.VE.1.2, E TO.VE.2.1, E IM.AC.9, E IM.MO.1.2, E IM.MO.3.2
Critical dependencies	E IG.AC.4
Actual subcontractor schedule performance	E SM.AC.7.2
Schedule status of managing subcontract activities	E SM.MO.1
Schedule for SQA activities	E QA.MO.1.1, E QA.MO.1.2
Schedule of activities for managing requirements	E RM.MO.1
Schedule status SCM activities	E CM.MO.1.1, E CM.MO.1.2
Schedule for process definition and improvement activities	E PF.MO.1.1
Schedule for process definition activities	E PD.MO.1.1
Schedule of intergroup coordination activities	E IG.MO.1.3, E IG.MO.1.4
Schedule for process measurement and analysis activities	E PA.MO.1
Schedule of defect prevention activities	E DP.MO.1.1
Schedule of technology innovation activities	E TI.MO.2
Software activity completion dates	E TO.AC.7.4
Identification of software products to be developed, including:	E PP.AC.7.4
Major products used by software engineering group	
Products used by other groups	
Products for external delivery	
Size estimates	E SM.AC.3.7.1
Size for software products and activities	E TO.AC.4.1
Total software size	E TO.AC.4.4

Progress (Continued)

Indicator	CMM Category
Actual size of Generated code Fully tested code Delivered code New Off-the-shelf Reused	E TO.AC.4.2, E IM.MO.1.1, E IM.AC.6.1, E IM.AC.6.6.1
Units of delivered documentation	E TO.AC.4.3
Software units tested	E TO.AC.7.2
Software units integrated	E TO.AC.7.2
Completion dates of test executions	E TO.AC.7.3
System release contents	E TO.AC.8.2
Cost estimates	E SM.AC.3.7.2
Total number of configuration items	E CM.AC.5
Number of configuration items by type	E CM.AC.5.1
Number of courses offered	E TR.MO.1.2
Number of process changes	E PC.AC.9.2.1, E PC.AC.9.2.5
Product and process metrics	E PA.AC.3.5, E TI.AB.3

Effort

Indicator	CMM Category
Staff resource estimates by	E PP.AC.7.6, E PP.AC.10.3.3,
Life-cycle stage	E IM.MO.3.1
Task	
Spread over time	
Skill level	E PP.AC.10.3.4
Effort and staffing	E TO.AC.5.3, E TO.MO.1.3, E TO.VE.1.2, E TO.VE.2.1
Actual subcontractor effort	E SM.AC.7.2
Effort for project planning activities	E PP.MO.1
Effort expended for SQA activities	E QA.MO.1.2
Effort for SCM activities	E CM.MO.1.2
Effort expended for process definition and improvement activities	E PF.MO.1.1
Effort expended for process definition activities	I PD.MO.1
Effort of intergroup coordination activities	E IG.MO.1.1, E IG.MO.1.2
Effort spent on defect prevention activities	E DP.MO.1.2
Effort of process improvement activities	E PC.AC.7
Total number of changes, over time	E PE.MO.2.3
Proposed	
Open	
Approved	
Incorporated into baseline	
Product and process metrics	E PA.AC.3.5, E TI.AB.3

Cost

Indicator	CMM Category
Cost estimates by Life-cycle stage Task Spread over time	E PP.AC.10.3.3,
Actual costs over time	E IM.AC.7, E IM.MO.1.3 E TO.AC.5.1, E TO.MO.1.3, E TO.VE.1.2, E TO.VE.2.1
Software costs per element	E TO.AC.5.2, E TO. MO.1.2
Actual subcontractor costs	E SM.AC.7.2
Cost for managing requirements	E RM.MO.1
Cost for project planning activities	E PP.MO.1
Cost for managing subcontract	E SM.MO.1
Cost of SQA activities	E QA.MO.1.2
Cost for SCM activities	E CM.MO.1.2
Cost for process definition and improvement activities	E PF.MO.1.1
Cost for process definition activities	E PD.MO.1.2
Cost of intergroup coordination activities	E IG.MO.1.1, E IG.MO.1.2
Cost of process measurement and analysis activities	E PA.MO.1
Cost of achieving quality goals	E QM.AC.11.1, E QM.MO.1.2
Cost of quality	E QM.MO.1.1
Cost of defect prevention activities	E DP.MO.1.1, E DP MO.1.2, E DP.VE.1.5, E DP.VE.1.6
Cost of technology innovation activities	E TI.MO.1, E TI.MO.2
Cost of process improvement activities	E PC.AC.7
Total number of changes, over time Proposed Open Approved Incorporated into baseline	E PE.MO.2.3
Product and process metrics	E PA.AC.3.5, E TI.AB.3

Software Quality Assurance Audit Results

Indicator	CMM Category
Number of SQA reviews	E RM.VE.3, E PP.VE.2, E TO.VE.3, E TR.VE.4, E SM.AC.10.2, E SM.VE.3, E QA.AC.2.5, E QA.AC.4, E QA.MO.1.3, E PD.VE.1, E CM.CO.1.5, E CM.VE.3, E CM.AC.7.3, E CM.VE.4, E IM.VE.3, E PE.VE.3, E IG.VE.3, E PR.VE.1, E PA.VE.5 , E QM.VE.5, E DP.VE.2, E TI.VE.2, E PC.VE.2
Number of baseline audits	E CM.AC.11, E CM.VE.4
Number of SQA audits	E RM.VE.3, E PP.VE.2, E TO.VE.3, E SM.AC.11.3, E SM.VE.3, E QA.MO.1.3, E QA.AC.2.5, E QA.AC.4, E CM.VE.4, E PD.VE.1, E TR.VE.4, E IM.VE.3, E PE.VE.3, E IG.VE.3, E PR.VE.1, E DP.VE.2, E PA.VE.5, E QM.VE.5, E TI.VE.2, E PC.VE.2
Number of deviations and product deficiencies Open Closed Type	I RM.VE.3, I PP.VE.2, I TO.VE.3, I SM.VE.3, I SM.AC.10.2, I SM.AC.11.3, E QA.AC.4.2, E QA.AC.5.3, I CM.AC.7.3, I CM.VE.3, I CM.VE.4, I CM.AC.11, I PD.VE.1, I TR.VE.4, I IM.VE.3, I PE.VE.3, I PR.VE.1, I IG.VE.3, I PA.VE.5, I QM.VE.5, I DP.VE.2, I TI.VE.2, I PC.VE.2

Review Results

Indicator	CMM Category
Number of action items	E TO.AC.12.6, E SM.AC.9.3,
Open	I IM.AC.3
Closed	
Type	
Priority	

Trouble Reports

Indicator	CMM Category
Number of trouble reports Open Approved Closed	E TO.AC.8.3, E CM.AC.6, E PE.MO.2.2, E PE.VE.3.9
Total number of defects detected	E PE.AC.10, E PE.MO.1.1, E DP.AC.2.2
Number of defects	
by type/category	E PE.AC.10.2, E PE.MO.1.1, E DP.AC.2.3, E DP.VE.1.1
by severity	E PE.AC.10.3, E PE.MO.1.1
by life-cycle stage	E PE.MO.1.1, E DP AC.7.1, E DP.MO.2
by activity introduced	E PE.AC.10.6, E DP AC.7.1, E DP.MO.2
Number of units affected by defect	E PE.AC.10.5
Number of units containing defect	E PE.AC.10.4

Peer Review Results

Indicator	CMM Category
Number of peer reviews	I PR.AC.1
Number of re-reviews	I PR.AC.3, E PR.MO.1
Action items	E PR.AC.2.5, E PR.AC.3.6
Open	
Closed	
Number of items reviewed at meeting	I PR.AC.3.1
Product size	E PR.AC.3.2, E PR.MO.1.3
Preparation lead time	E PR.MO.1.1
Preparation time for each reviewer	E PR.AC.3.3, E PR.MO.1.2
Length of time of review	E PR.AC.3.4, E PR.MO.1.6
Size of review team	I PR.AC.1.4, E PR.MO.1.4
Experience of review team	E PR.AC.1.4, E PR.MO.1.5
Structure of review team	E PR.AC.1.4
Number of defects detected	E PR.AC.3.5, E IM.MO.1.4, E PE.AC.10, E PE.MO.1.1, E DP.AC.2.2
Number of defects	
by type/category	E PE.AC.10.2, E PE.MO.1.1, E DP.AC.2.3, E DP.VE.1.1
by severity	E PE.AC.10.3, E PE.MO.1.1
by life-cycle stage	E PE.MO.1.1, E DP AC.7.1, E DP.MO.2
by activity introduced	E PE.AC.10.6, E DP AC.7.1, E DP.MO.2
Number of defects identified by type	E PR.AC.3.5
Number of units affected by defect	E PE.AC.10.5
Number of units containing defect	E PC.AC.10.4
Rework effort	E PR.AC.3.7, E PR.MO.1.8